3-8-2020

Effective Algorithms for the Closest Pair and Related Problems

Xingyu Cai
University of Connecticut - Storrs, xingyu.cai@uconn.edu

Effective Algorithms for the Closest Pair and Related Problems

Xingyu Cai, Ph.D.

University of Connecticut, 2020

ABSTRACT

The Closest Pair problem aims to identify the closest pair (using some similarity measure, e.g., Euclidean distance, Dynamic Time Warping distance, etc.) of points in a metric space. This is one of the fundamental problems that has a wide range of applications in the data mining area, since most of the data can be represented in a vector form residing in a high dimensional space, and we would like to identify the relationship among those data points. Typical applications include but not limited to, social data analysis, user pattern identification, motif mining in biological data, data clustering, etc. This is a very classical problem and has been studied very well in the past decades.

In this thesis, we study the Closest Pair problem and its variants, and also bring the machine learning perspective to solve some closely related problems. In particular, we have proposed two approximate algorithms to efficiently address the Closest Pair of Points (CPP) problem, and one deterministic approach to solve the Closest Pair of Subsequences (CPS) problem, using Euclidean distance measure. In addition, to identify the closest subsequences in the time series data, we have proposed a learnable feature extractor embedded in an artificial neural network, to learn patterns in the

scope of the Dynamic Time Warping metric. In the end, to speed up the inference speed of the proposed algorithm, we have also proposed a neural network pruning technique to obtain a smaller network with similar capacity.

All the proposed methods are shown to have achieved the state-of-the-art performance in various standard benchmark datasets.

Effective Algorithms for the Closest Pair and Related Problems

Xingyu Cai

M.S., Xi'an Jiaotong University, 2012

B.S., Xi'an Jiaotong University, 2010

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2020

APPROVAL PAGE

Doctor of Philosophy Dissertation

Effective Algorithms for the Closest Pair and Related Problems

Presented by

Xingyu Cai, B.S., M.S.

Major Advisor _____
Dr. Sanguthevar Rajasekaran

Associate Advisor _____
Dr. Yufeng Wu

Associate Advisor _____
Dr. Song Han

University of Connecticut

2020

ACKNOWLEDGMENTS

Foremost, thank you to my major advisor, Dr. Sanguthevar Rajasekaran for his continuous support of my PhD study and research. His broad knowledge and enthusiasm, his persistent searching for the truth of science have deeply influenced me. His patience, motivation and guidance helped me in all the time of my research and the writing of this thesis. He is both an advisor in my PhD research, as well as a mentor in my life.

Besides my major advisor, I would also like to express my sincere gratitude to the rest of my committee: Dr. Yufeng Wu and Dr. Song Han. Their valuable comments and encouragement inspired me and helped me a lot in finishing this thesis.

My deepest appreciation also goes to all the faculty and staff in the CSE department for their support during my PhD. In particular, I thank Prof. Zhijie Shi for his support in my early research.

I would like to take this opportunity to thank my labmates and co-authors, making my time at UConn meaningful and enjoyable.

Lastly, I am forever indebted to my family, for their unconditioned love and tolerance throughout my life. Thank Shanglin Zhou for her accompany to accomplish all these. This thesis would not have been possible without their caring and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview and Background

In data mining and machine learning areas, similarity search is one of the core techniques that drive a large variety of applications. In fact, almost all the ending tasks such as fraud identification, social relationship analysis, etc., heavily rely on features extracted from similarity search. These tasks could benefit tremendously from a fast and accurate search based on some similarity measure.

Typically there are two key stages when performing the similarity identification. The first is to define a proper similarity or distance measure for the application. Sometimes the measure itself could be learned from the data, which is a research topic named as metric learning. In other cases, we need to consider the characteristics of the data and define the proper metric. In this thesis, we cover commonly used metric such as Euclidean distance, Manhattan distance and the non-linear Dynamic Time Warping distance.

The second stage is to retrieve similar objects or features in the scope of the defined distance measure. Typical techniques to address this problem include Nearest Neighbor (NN) and finding the Closest Pair (CP). These two problems are closely related. For instance, the CP could be seen as an extension of the NN, which requires more computation and thus is more challenging. The goal of this thesis is to study the CP problem and provide new algorithms that could benefits the communities.

In this thesis, we first study the Closest Pair of Points (CPP) problem. The early effort to address the CPP problem could trace back to 1979 with a run time of $O(N \log \log N)$, assuming that the floor operation takes $O(1)$ time and the dimension is constant [24]. A lot of follow up works have been done, particularly pushing the research to high dimensional cases, where the dimension number m cannot be ignored. Along this line, Mueen, et al., have presented an exact algorithm called MK [57], though the MK algorithm is originally proposed to solve a special case of the CPP problem, namely the Time Series Motif Mining (TSMM) problem. There are more advanced TSMM solvers now, e.g. [87], but MK is still one of the best-performing published exact algorithms that solve the CPP in practice.

The second problem we are targeting in this thesis is the Closest Pair of Subsequences (CPS) problem. This is a special version of the CPP problem but work with the sequence data. The sequence data could be numerical sequences or biological sequences. We use Euclidean distance and Hamming distance for the CPS problem. An upper bound of the running time could be obtained from algorithms designed for the CPP problem, such as [87, 44]. In another line of this research, the Longest Common Substring (LCS) problem could be viewed as a dual version of CPS. While CPS identifies a pair of substring (of length l) that have the minimum distance for a given l, the LCS tries to find the length l of the longest-common-substring in the

given sequences [28, 41]. For the CPS problem, there are deterministic algorithms [66], as well as approximate algorithms such as [2, 52]. In this thesis we focus on the deterministic approaches.

In the domain of time series, both Euclidean distance or Manhattan distances fail to serve as a proper similarity measure. This is due to the warping effect (scaling and shifting in the time axis). To overcome the warping effect, Dynamic Time Warping (DTW) [4] has been widely used. As a matter of fact, the finite and discrete version of DTW is known as the Edit distance and being well studied in [27]. To identify the closest pair of sequences in the scope of warping cases, and further achieve similarity search and pattern mining for time series data, we exploit the properties of DTW computation, and propose a novel learning framework to extract patterns in the time series. This framework improves the conventional approach using predefined DTW features [36] by providing a learnable feature extractor, thus become more flexible and can be integrated into other deep learning frameworks to perform different tasks.

It is a increasing need to speed up the computation of the above algorithms. In particular, the learnable DTW feature extractor embedded in an artificial neural network requires very large computation power. Thereafter we propose a deep neural network pruning method to reduce the network size and reduce the inference time. Note that this is a general approach that works on most modern network architectures. Compared to other pruning methods like [30, 82], we incorporate adversarial attack techniques [75] to better guide us pruning the target components.

1.2 Outline of the Dissertation

This dissertation is organized as follows: In Chapter 2, we target the Closest Pair of Points (CPP) problem, and proposed two approximate algorithms that both achieve the state-of-the-art performances. Next, in Chapter 3, we focus on a very similar problem named the Closest Pair of Subsequences (CPS) problem. This is a variant of the original CPP problem but applied on the sequence data, thus we can exploit some advantages that only exists in the sequence data. We proposed a deterministic algorithm to solve the CPS problem and obtain significant speedups over the existing approach.

Both the above CPP and CPS problems are defined and addressed in the scope of Euclidean distance metric. However, in the time series domain, another very useful metric called Dynamic Time Warping is widely adopted. To identify the closest pattern in the time series data, we proposed a learnable feature extractor (kernel) embedded in an artificial neural network in Chapter 4. On top of that, we show this learnable kernel could also be used in different tasks and achieve very promising results. Last but not least, to speed up our neural network inference speed, we proposed a network pruning framework in Chapter 5, which is shown to be effective in several modern network architectures.

We draw the conclusion, as well as discussing the research impact and potential future work of the proposed approaches above in Chapter 6.

1.3 Publications

Conference papers that are accepted and published with primary authorship include [10, 9, 7, 11, 5]:

1. **X. Cai**, J. Yi, F. Zhang, and S. Rajasekaran, "Adversarial structured neural network pruning," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 2433–2436.

2. **X. Cai**, T. Xu, J. Yi, J. Huang, and S. Rajasekaran, "DTWNet: a dynamic time warping network," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 636–11 646.

3. **X. Cai**, S. Rajasekaran, and F. Zhang, "Efficient approximate algorithms for the closest pair problem in high dimensional spaces," in *Pacific-Asia Conference onKnowledge Discovery and Data Mining*. Springer, 2018, pp. 151–163.

4. **X. Cai**, S. Zhou, and S. Rajasekaran, "Jump: a fast deterministic algorithm to find the closest pair of subsequences," in *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 2018, pp. 73–80.

5. **X. Cai**, A.-A. Mamun, and S. Rajasekaran, "Novel algorithms for finding the closest l-mers in biological data," in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2017, pp. 525–528.

Journal papers that are accepted and published with primary authorship include [6, 50, 8]:

1. **X. Cai**, A.-A. Mamun, and S. Rajasekaran, "Efficient algorithms for finding the closest l-mers in biological data," *IEEE/ACM transactions on computational*

biology and bioinformatics, vol. 16, no. 6, pp. 1912–1921, 2018.

2. K. F. Lyon, **X. Cai**, R. J. Young, A.-A. Mamun, S. Rajasekaran, and M. R. Schiller, "Minimotif miner 4: a million peptide minimotifs and counting," *Nucleic acids research*, vol. 46, no. D1, pp. D465–D470, 2018.

3. **X. Cai**, L. Wan, Y. Huang, S. Zhou, and Z. Shi, "Further results on multicarrier mfsk based underwater acoustic communications," *Physical Communication*, vol. 18, pp. 15–27, 2016.

Conference papers that are accepted and published with co-authorship include [81, 59, 83, 63, 62, 68]:

1. Z. Wang, A.-A. Mamun, **X. Cai**, N. Ravishanker, and S. Rajasekaran, "Efficient sequential and parallel algorithms for estimating higher order spectra," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 1743–1752.

2. P. Xiao, **X. Cai**, and S. Rajasekaran, "Efficient algorithms for finding edit-distance based motifs," in *International Conference on Algorithms for Computational Biology*. Springer, 2019, pp. 212–223.

3. P. Xiao, **X. Cai**, and S. Rajasekaran, "Ems3: An improved algorithm for finding edit-distance based motifs," in *2018 IEEE 8th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2018, pp. 1–1.

4. S. Pathak, **X. Cai**, and S. Rajasekaran, "Ensemble deep timenet: An ensemble learning approach with deep neural networks for time series," in *2018 IEEE*

8th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS). IEEE, 2018, pp. 1–1.

5. S. Pathak and **X. Cai**, "Ensemble learning algorithm for drug-target interaction prediction," in *2017 IEEE 7th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2017, pp. 1–1.

6. S. Rajasekaran, S. Saha, and **X. Cai**, "Novel exact and approximate algorithms for the closest pair problem," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 1045–1050.

Chapter 2

The Closest Pair of Points Problem

2.1 Introduction

Similarity search has been widely used in data mining. Example applications include finding the similarity between user patterns from online merchant transactions, analysis of social media connections, unsupervised data clustering, knowledge discovery from semantic data, etc. Two of the fundamental problems in data mining are finding the Nearest Neighbor (NN) and finding the Closest Pair (CP). These two problems are closely related. For instance, CP could be seen as an extension of NN, which requires more computation and thus is more challenging. In general, multi-feature data could be modeled as points in a high dimensional metric space. Among all the different similarity measurement metric, ℓ_p norm is commonly used. In this chapter, ℓ_2 norm, or Euclidean distance, is employed as it is one of the most widely applicable measurements.

The Closest Pair Problem (CPP) we are addressing is that of identifying the clos-

est pair of points from a given set of N points $\in \mathcal{R}^m$ when m is not small. This classical problem has been studied extensively [71]. A straightforward algorithm for solving this problem takes $O(N^2m)$ time, where m is the dimension of the input space. Research works have been carried out in different domains for different purposes to solve this problem in an efficient way. In 1979, Fortune and Hopcroft presented a deterministic algorithm with a run time of $O(N \log \log N)$ assuming that the floor operation takes $O(1)$ time [24]. In [3], another divide-and-conquer deterministic algorithm was introduced. Later improvements include [67, 25, 34, 64]. In his seminal paper, Rabin proposed a randomized algorithm with an expected run time of $O(N)$ [22] (where the expectation is in the space of all possible outcomes of coin flips made in the algorithm). Rabin's algorithm also used the floor function as a basic operation. In 1995, the sieve method was proposed to eliminate points in a randomized way such that the actual comparison of the remaining candidates could be dramatically reduced [38]. A sample-based randomized approach was proposed in [21] in 1997 to solve several issues existing in [22]. Yao has proven a lower bound of $\Omega(N \log N)$ on the algebraic decision tree model (for any dimension) [84]. All these algorithms assume a constant dimensional space (i.e., $m = O(1)$), and the run times are exponentially dependent on the dimension, making them not applicable for a dimension of several hundreds.

In recent years, database applications have driven the research on CPP. By exploring the connection between CPP and matrix multiplication, Indyk [33] has presented an $O(N^{(w+3)/2)}$ time algorithm for CPP in ℓ_1 and ℓ_∞ norms, where $O(N^w)$ is the time needed to multiply two $N \times N$ matrices. This algorithm is not applicable for ℓ_2 norm. Corral et al. [16] have provided a method that uses tree data structures. Besides exact algorithms, Lopez et al. [47] have provided an approximate algorithm to address

this problem by making copies of the original data and employing random shifting on each copy. More recently, Locally Sensitivity Hashing (LSH) has gained attention in solving the NN problem. As a consequence, approximate algorithms based on LSH for CPP are proposed in the literature. Datar [18] has proposed a sub-quadratic time algorithm using LSH that solves the c-approximate problem (output neighbors that are no further than c times the distance between the nearest neighbors). Later Tao [76] improved Datar's algorithm and extended it to out-of-core CPP, where the I/O costs are optimized. The comparison in [76] shows that their algorithm outperforms the methods in [47, 16]. These algorithms mainly focus on the NN problem, or address the problem for efficiency in I/O cost, making them fit for out-of-core computation with many applications such as in database query processing. However, they are not very suitable for in-memory computation. Also, the construction of special data structures (such as LSB tree in [76]) will bring significant overhead for in-core tasks. In addition, the approximate methods in this domain are addressing the c-approximate problem that introduces a relaxation factor c.

Mueen, et al., have presented an elegant exact algorithm called MK for the CPP [57]. Though this algorithm was originally proposed to solve a special case of the CPP, known as the time series motif mining problem, it can be used to solve the CPP very well. Although MK is an $O(N^2m)$ time algorithm, it improves the performance of the brute-force algorithm in practice using the triangular inequality and a technique called early-abandoning. MK is a deterministic algorithm that always finds the closest pair. To the best of our knowledge, MK is still one of the best performing algorithms for high dimensional CPP in practice, even though it is no longer the state-of-the-art choice for time series motif mining problem. In this chapter we use MK as the baseline to evaluate our proposed algorithms. To pro-

vide a fair comparison, we use the original MK code that is publicly available in http://alumni.cs.ucr.edu/mueen/MK/. The code for our proposed approaches can be found at https://github.com/TideDancer/ACPP.git.

In this chapter we present two approximate algorithms for the CPP. One of them revisits the divide-and-conquer approach but modifies it to high dimensional settings. The other uses a novel idea in random projection: The original Johnson-Lindenstrauss Lemma shows the existence of a random projection of $O(\log N)$ dimension that preserves all pairwise distances with a high probability. For the CPP we only have to preserve the distance between the closest pair. We use random projection of points into 1D. We show that if we perform this projection $O(\log N)$ times, then the distance between the closest pair will be preserved at least once with a high probability. Note that although the proposed algorithms are sequential, all these algorithms along with MK, could be easily parallelized due to the independence of their subroutines.

The rest of this chapter is organized as follows: In Section 2.2, we present two approximate algorithms for the high dimensional CPP. Running time and accuracy bounds are proved for both of the approaches. Comprehensive experiments are carried out to evaluate the performance of both algorithms in Section 2.3, and some conclusions are provided at the end.

2.2 Proposed Approximate Algorithms

Two approximate approaches for the high dimensional CPP are provided: ACP-P and ACP-D. Both algorithms always keep an upper bound δ_u on the distance δ^* between the closest pair of points. The common initial step for both algorithms is to obtain

an upper bound on δ^* by picking a random sample of $\sqrt{N}$ points and identifying the distance between the closest pair of points in the sample. Even a brute-force algorithm will only take $O(N)$ time for doing this.

2.2.1 ACP-D

The divide-and-conquer algorithm of [3] performs well on low dimensional (e.g., 2D and 3D) data with a run time of $O(N \log N)$. Its performance degrades significantly on high dimensional data since the run time has an exponential dependence on the dimension. The divide-and-conquer algorithm proceeds by partitioning the input into two using the median along one of the dimensions. The closest pairs are recursively found for each of the two parts. Followed by this, we have to find the closest among the cross-part pairs. When the input is from 2D (i.e., $m = 2$), the number of cross-pairs that have to be considered is proved to be $O(N)$. When the input is from an m-dimensional space, the number of candidate cross-pairs to be considered goes up to $O(N \times 3^m)$. This number can be $\Omega(N^2)$ or worse. Thus the performance could be as bad as that of the brute force algorithm.

In this section we propose an enhanced divide-and-conquer algorithm. The idea is to choose the candidate cross-part pairs appropriately. Here again we partition the input into two and recursively find the closest pair in each part. To find the closest cross-part pair we do the following: Let $\mathcal{H}$ be the hyperplane that partitions the input into two (based on the median along one of the coordinates). We have to consider all pairs of the form (a, b) where a is one side of the hyperplane $\mathcal{H}$ and b is on the other side of $\mathcal{H}$. Instead of checking all such pairs we only consider pairs where a and b are on different sides of $\mathcal{H}$ but within a distance of s. We refer to s as

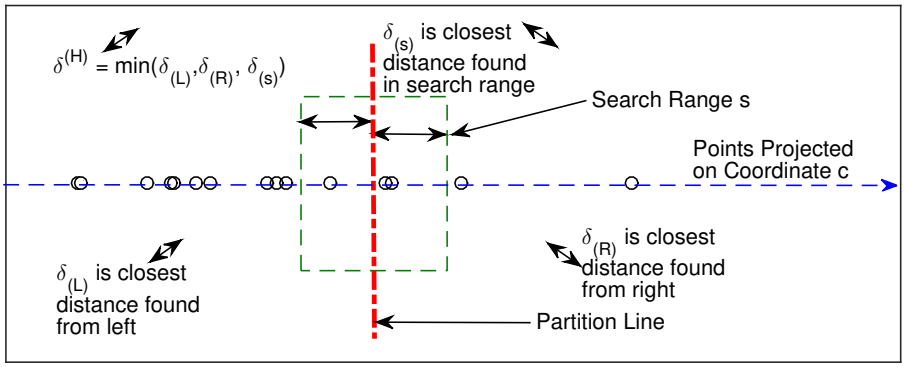


FIGURE 2.1: Illustration of search within a range s around the partition line along a particular coordinate

Algorithm 1 ACP-D

Input: N points $p_i \in \mathcal{R}^m$ $(1 \leq i \leq N)$, brute-force subset size T, search range constant α. Initialize left = 1, right = N, depth = 1

Output: function ACP-D(depth, left, right) finds the closest pair (l, r) with the smallest Euclidean distance $D(l, r)$

len = right - left + 1
if depth = 1 **then**
  Randomly select one coordinate $c^{(\mathcal{H})}$, $c^{(\mathcal{H})} \in [1, d]$;
  Sort the points based on values $p_i[c^{(\mathcal{H})}]$ along the coordinate $c^{(\mathcal{H})}$;
end if
if len $\leq T$ **then**
  Use brute-force to find best-so-far $d(i, j)$ where left $\leq i \leq j \leq$ right;
  if $d(i, j) < D(l, r)$ **then**
    $l = i$; $r = j$; $D(l, r) = d(i, j)$;
  end if
  return $D(l, r)$;
else
  mid = left + len/2;
  ACP-D(depth+1, left, mid);
  ACP-D(depth+1, mid+1, right);
  Obtain two sets of indices: $S = \{p_i\}, i < \text{mid}, p_{mid} - p_i \leq \alpha D/\sqrt{N}$ and $S' = \{p_j\}, j > \text{mid}, p_j - p_{mid} \leq \alpha D/\sqrt{N}$;
  for $i \in S$ **do**
    for $j \in S'$ **do**
      if dist$(p_i, p_j) < D(l, r)$ **then**
        $l = i$; $r = j$; $D(l, r) =$ dist(p_i, p_j);
      end if
    end for
  end for
end if

the search range (see Figure 2.1). This procedure is repeated by partitioning along different coordinates to increase the chances of finding the closest pair.

To begin with, ACP-D randomly chooses a coordinate to do partition. It then recursively finds the closest pair's distance from the left and the right partitions (denote the distances as $\delta_{(L)}, \delta_{(R)}$). Next we look at all the points that reside within a search range s around the partition hyperplane $\mathcal{H}$ along this coordinate, and find the closest pair (distance as $\delta_{(s)}$) among these candidates. Followed by this we update the pair with the minimum distance denoted as $\delta^{(\mathcal{H})} = \min(\delta_{(L)}, \delta_{(R)}, \delta_{(s)})$. The detailed pseudocode of ACP-D is given in Algorithm 1. An illustration of searching within a range s is shown in Figure 2.1. We establish the following theorem. This theorem offers a probabilistic bound on success rate and run time of ACP-D. To boost the success rate, we repeat ACP-D and output the closest pair seen as the closest pair of points.

Theorem 2.2.1. *Let $p[c]$ represent vector p's c-th element, or equivalently p's c-coordinate value. Assume that the coordinate values in each dimension are uniformly distributed and let the spread length of points be $r = \max_i p_i[c^{(\mathcal{H})}] - \min_j p_j[c^{(\mathcal{H})}]$ on the partition coordinate $c^{(\mathcal{H})}$. Use a search range of $s = \sqrt{\alpha}\frac{\delta^{(\mathcal{H})}}{\sqrt{m}}$. As long as $r^2 = \Omega(N)$ where m is the dimension, ACP-D algorithm's expected run time will be $T(N) = O(N \log N)$, with a high probability.*

Corollary 2.2.2. *We have the following probability bound on the run time:*

$$Prob\{T(N) \geq (\beta+1)\alpha(\delta^{(\mathcal{H})})^2 N \log N\} \leq e^{-\beta}, \text{ for any } \beta > 0.$$

2.2.2 ACP-P

Random projection lemma [35] states that pairwise distances are closely preserved in a random $O(\log N)$-dimensional space with a high probability. In this chapter we prove that, if we repeat projecting the input points from $\mathcal{R}^m$ to $\mathcal{R}^d$ randomly ($d < m$) a total of k times, as long as kd satisfies a certain condition, the closest pair's distance will be closely preserved in at least one of the projections, with a high probability. In addition, $d = 1$ would significantly reduce the computation cost. We exploit this property in the ACP-P algorithm.

After the projection, all the pairs in the projected space that are within a distance of $\delta^{(P)} = (1 + \epsilon)\delta_u$ (in $\mathcal{R}^d$) needs to be identified, where ϵ is a small constant. For the case of $d > 1$, identifying these close pairs in $\mathcal{R}^d$ still remains a difficult task. One can use hyper-sphere centered at each point with a radius of $\delta^{(P)}$, and check if there are other points in the hyper-sphere. However, this might be even harder than directly computing all pairwise distances in $\mathcal{R}^d$, which takes $O(N^2 d)$ running time. On the other hand, if in 1-D space ($d = 1$), the hyper-sphere becomes left and right intervals, making the job of identifying close points within an interval of $\delta^{(P)}$ extremely easy. To be specific, one can use any sorting algorithm to first sort all the projected points because all the points are identified by a scalar value in 1-D space. Then a scanning from left to right is performed and all the adjacent points within a certain range are detected. In total it only requires an $O(N \log N)$ running time. In fact, sorting could be replaced by a griding approach to identify pairs within an interval. After identifying these pairs, the Euclidean distance between each pair is computed in $\mathcal{R}^m$. The pair with the least distance is kept and δ_u is updated.

The above projection-identification process is repeated k times. We show that if

Algorithm 2 ACP-P

Input: N points in $\mathcal{R}^m$: $p_1, p_2, \ldots, p_N$.
Output: The closest pair of input points.

1: $j = 1$
2: **repeat**
3: Randomly generate a projection vector $\Phi \in \mathcal{R}^{1 \times n}$
4: **for** $i = 1$ to N **do**
5: $p'_i = \Phi p_i^T$
6: **end for**
7: Sort $p'_1, p'_2, \ldots, p'_N$;
8: **for** $i = 1$ to N **do**
9: Identify the interval that p'_i belongs to;
10: **end for**
11: **for** every interval **do**
12: Generate all possible pairs from the points that have fallen into this interval. These are candidate pairs;
13: **end for**
14: For each candidate pair compute the distance in $\mathcal{R}^m$ and pick the pair with the least distance. Let this distance be δ_j;
15: $j = j + 1$;
16: **until** $j = k$
17: Find $\delta_o = \min\{\delta_1, \delta_2, \ldots, \delta_k\}$;
18: **return** δ_o

$kd = \Theta(\log N)$, then the closest pair would come within a distance of $(1+\epsilon)\delta_u$ in the projected space at least once with a high probability. Note that in the original Johnson-Lindenstrauss Lemma, $d = O(\log N)$. **The reason that we can push the limit to $d = 1$ is because we only have to preserve the distance between the closest pair, and not all pairwise distances.** We provide the following theorems and the corresponding proofs.

Theorem 2.2.3. *Let the closest pair have a distance of δ^*. If we repeat the random projection $\mathcal{R}^m \to \mathcal{R}^d$ for a total of k times, then the probability that $(\delta^{(P)})^2 < (1+\epsilon)(\delta^*)^2$ at least once is high (i.e., $\geq 1 - N^{-\alpha}$ where α is some constant), as long as $dk \geq \frac{4\alpha}{\epsilon^2 - \epsilon^3} \log N$, for any $\epsilon \in [0, 1]$.*

Corollary 2.2.4. *Let $d = 1$. In each iteration of ACP-P, let the projected points be quantized with intervals of length $2(1+\epsilon)\delta_u$. The probability that the closest pair (in $\mathcal{R}^m$) will fall into the same interval is $\geq \frac{1}{2}[1 - e^{-(\epsilon^2 - \epsilon^3)/4}]$. This in turn means that*

the number of iterations taken by ACP-P to identify the closest pair of points with a high probability is $k = O\left(\frac{\alpha \log N}{\epsilon^2 - \epsilon^3}\right)$.

In practice, we have found that the sorting based implementation in 1-D does not introduce an observable overhead. A detailed pseudocode of ACP-P that employs sorting is given in Algorithm 2. It is worth pointing out that the key difference between ACP-P and the LSH method used in [76] is after projection. The linear search based on the sorted list of points is much more efficient to identify each points' close neighbors, rather than a grid scheme using hashset technique. In our experiments we have realized that neither C++/boost hashset nor google's hashset could achieve desirable in-core performance, making the method in [76] not suitable for in-memory computations.

2.3 Experiments

We have conducted experiments to evaluate the performance of ACP-P and ACP-D against MK. We have employed an Intel Xeon E5 CPU @ 3.2 GHz machine. The experiments have been performed on synthetic datasets, with different numbers of points and dimensions. Coordinate values have been generated uniformly randomly from the range: $[0, 1000]$. The following values have been used: N = 10, 20, 30, 40, 50 $\times 10^3$ and $m = 128, 256, 512, 1024$ and 2048.

To further boost the success rate, in each run we repeat the approximate algorithms Q times and output the best among them. Q is designed as $Q_{\text{ACP-D}} = h\frac{N}{10\times 10^3}$ and $Q_{\text{ACP-P}} = h(\frac{N}{10\times 10^3})^2$ for ACP-D and ACP-P, respectively. N is the input size and h is the hyper parameter. For instance if $N = 30k, h = 2$, then $Q = 6$ for ACP-D

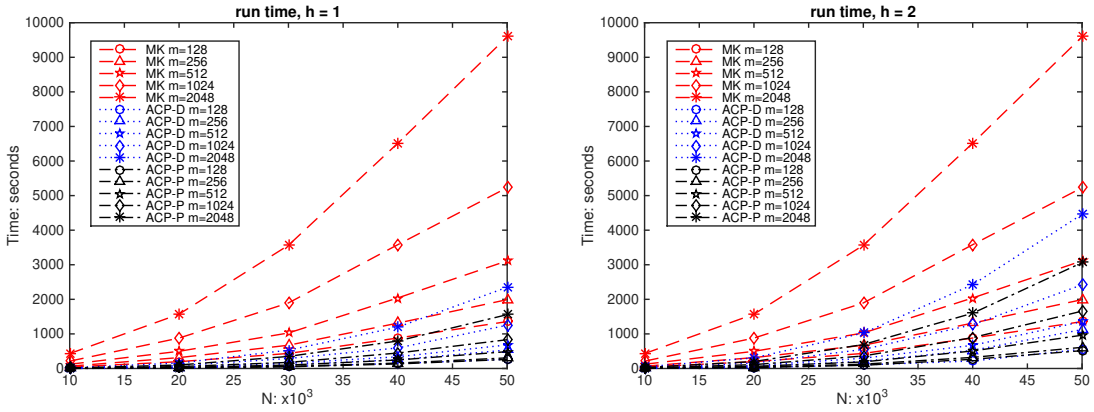


FIGURE 2.2: Run time comparison

and $Q = 18$ for ACP-P. We perform 10 runs and provide the average running time, average rank and the hit rate (i.e., the fraction of the number of times the closest pair is found in 10 runs). Clearly, the larger the h, the better is the accuracy and the worse is the run time.

Figure 2.2 shows the run time comparison. Clearly, for all settings, ACP-D and ACP-P are significantly faster than the MK algorithm. As expected, the run time when $h = 2$ (the right plot) is longer than when $h = 1$ (the left plot) for both approximate algorithms. When N is smaller, ACP-D could be slightly faster, but when N is larger, ACP-P becomes the fastest. For instance, when $N = 50k, m = 1,024, h = 1$, ACP-D's run time is 1,252 seconds and ACP-P's is 832 seconds, while MK is much slower using 5,230 seconds.

To illustrate the accuracy, in Figure 2.3, the hit rate is presented. Again in the case of $h = 2$, the overall hit rate is higher as expected. When m is higher, the hit rate tends to be better than in smaller dimension cases. The average rank is also provided in Table 2.1. From the table we can see that for larger N, the proposed algorithms are more robust because the average ranks are closer to 1. And the overall

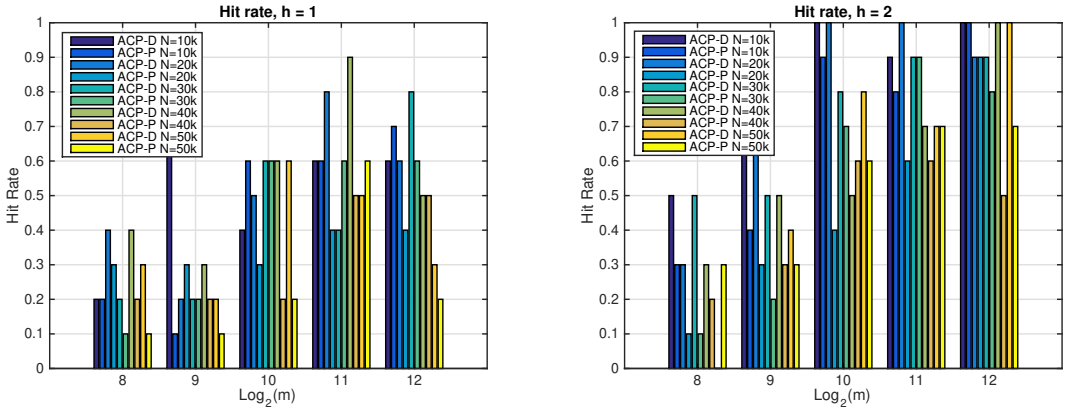


FIGURE 2.3: Hit rate comparison

TABLE 2.1: Average Rank (the smaller the better)

	$m = 128$		$m = 256$		$m = 512$		$m = 1024$		$m = 2048$	
$h = 1$	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P
N=10k	2.7	7.1	4	6.2	4.4	8.5	4	6.6	3.9	7.9
N=20k	1.7	4.7	2	3.7	3.5	4.9	3.6	5	3.2	5.5
N=30k	2.1	1.9	1.7	2.7	1.9	2.3	1.5	3.2	1.8	2.9
N=40k	1.5	1.9	1.2	3.3	2.4	2	1.3	2.9	2.1	2.2
N=50k	1.6	1.5	1.4	2.6	1.2	1.4	1.7	2	2	3.5
$h = 2$	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P
N=10k	1.6	4.8	2.5	3.7	1.7	5.5	2.2	5.8	3.5	5.7
N=20k	1.5	1.9	1.2	3.3	1.8	2.4	1.5	4.6	2.2	3.1
N=30k	1	1.2	1	2.1	1.2	1.5	1.5	2	1.3	1.6
N=40k	1.1	1.2	1	1.4	1.1	1.1	1.3	1.7	1.3	1.3
N=50k	1	1	1.1	1.1	1.1	1.2	1	1.8	1	1.4

average rank for $h = 2$ is also better than that for $h = 1$.

In addition to the rank of the best pair identified, we also report the difference between the output pair's distance and the true closest pair's distance. We define the distance ratio as $\rho = d/d^*$, where d^* is the distance between the closest pair of points and d is the distance between the output pair of points. In Table 2.2, we show the mean and variance of ρ when $h = 2$, and demonstrate that for our synthetic dataset, the distance ratio is very close to 1 with a small variance. This also proves the robustness of our proposed approximate algorithms.

TABLE 2.2: Distance Ratio ρ (Mean and Std) when $h = 2$

$h = 2$	$m = 128$		$m = 256$		$m = 512$		$m = 1024$		$m = 2048$	
Mean	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P
N=10k	1.010	1.034	1.004	1.008	1.002	1.012	1.001	1.004	1.006	1.006
N=20k	1.008	1.015	1.000	1.002	1.002	1.004	1.001	1.003	1.005	1.007
N=30k	1.000	1.004	1.000	1.002	1.000	1.001	1.000	1.001	1.001	1.002
N=40k	1.001	1.003	1.000	1.012	1.000	1.000	1.001	1.001	1.001	1.001
N=50k	1.000	1.000	1.002	1.002	1.000	1.000	1.000	1.005	1.000	1.001
Std	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P	ACP-D	ACP-P
N=10k	0.011	0.029	0.004	0.003	0.002	0.006	0.001	0.002	0.002	0.004
N=20k	0.015	0.015	0.000	0.003	0.003	0.003	0.001	0.003	0.005	0.005
N=30k	0.000	0.011	0.000	0.002	0.001	0.002	0.000	0.001	0.002	0.003
N=40k	0.004	0.006	0.000	0.014	0.001	0.001	0.001	0.002	0.001	0.001
N=50k	0.000	0.000	0.007	0.007	0.001	0.001	0.000	0.005	0.000	0.001

2.4 Conclusions

In this chapter we have offered two approximate algorithms for solving the CPP. Both of them are based on the idea of converting high dimensional search into line search. We provide theoretical bounds on the run time and prove the accuracy of ACP-D. For ACP-P, we exploit random projections but push the limit to 1-D space because we only identify the closest pair rather than preserving all pairwise distances. A theoretical analysis is also provided. In the experiments, we perform comprehensive simulations to evaluate both the run time and the accuracy for the proposed approximate algorithms. The results reveal that our approach runs much faster than the state-of-the-art method while still keeping a very good accuracy. Our algorithms could be easily parallelized for further speedups.

Chapter 3

The Closest Pair of Subsequences Problem

3.1 Introduction

Pattern identification in sequence data has been well studied in the past decades. Algorithms for solving this problem fall under two categories: supervised pattern matching if a pattern is provided; unsupervised pattern recognition such as finding the most similar subsequences. Numerous sequence mining techniques of both kinds have been proposed to address this problem. Among them, the Closest pair of subsequences (CPS) problem has attracted a lot of attention. CPS is in the second category. Given a sequence of data $A = a_1, a_2, \ldots, a_n$ and an integer $l < n$, the CPS problem is to identify the closest pair of subsequences of length l each in A. Specifically, we have to identify two subsequences A_1 and A_2 (with $A_1 = (a_j, a_j + 1, \ldots, a_{j+l-1})$, $A_2 = (a_{i+j-1}, a_{i+j}, \ldots, a_{i+j+l-2})$) such that these subsequences have the least distance

between them, from among all such pairs. In general, the input data sequence could be an arbitrary real sequence like time series data, or limited alphabet size data such as genomic sequences. Correspondingly, the distance or similarity metric could be Euclidean distance, cosine similarity, Damerau—Levenshtein distance, etc. A minimum interval constraint, i.e., $(i - l) > L$ where L is an integer (> 0), is usually applied to ensure that the target pair does not overlap or is not too close to each other.

In this chapter we address the CPS problem under Euclidean as well Hamming distances. Euclidean distance requires a significant amount of computations (additions and multiplications, especially multiplication operations) that slows down the existing approaches. On the other hand, other fast approaches that handle Hamming distances could use bit operations. There are efficient implementations of bit operations in C++, Java or other modern languages. Bit operations (such as and, or, xor, etc) have been shown to offer great speedups for numerous problems (see e.g., [68]). However, these bit operations do not apply to Euclidean distance or cosine similarity calculations. Without loss of generality, we consider Euclidean distance in our proposed algorithms but note that our approach can be easily extended to metrics that involve heavy computations, especially floating point multiplications.

In the past several years the CPS problem has been well studied in the domain of Time Series Motif Mining (TSMM) [46]. TSMM is a crucial problem that finds the most similar patterns in an input sequence. TSMM can be thought of as a special case of the CPS problem. A straightforward method for solving the CPS as well as the TSMM problems would take $O(n^2 l)$ time. The ideas is to compute the distance between every pair of possible subsequences. Mueen, et al. have presented an exact algorithm called MK [57] for the TSMM problem. Using a novel application

of the triangular inequality and the early abandoning method, MK improves the performance of the straightforward algorithm by a large factor.

There are also probabilistic and approximate algorithms proposed for solving the TSMM and CPS problems (see e.g., [2, 52, 53]). Though the approximate algorithms might have an edge on run times, they are only able to find the target with a certain probability and no post-check is available in real applications in most cases. Thus in this chapter we focus on providing an ultra fast deterministic algorithm. By exploiting the overlapping parts of consecutive subsequences, [66] first proposed an $O(n^2)$ time algorithm for solving the CPS problem. Note that this run time is independent of the subsequence length l and hence this algorithm can be applied on high dimensional datasets. Subsequently, this idea has been adapted in the recent work of Matrix Profile [86, 87] for the TSMM and the CPS problems. However, Matrix Profile [87] has to obtain all pairwise distances which would be a waste if we are only interested in the closest one. More importantly, all subsequent algorithms essentially utilize the $O(n^2)$ algorithm proposed in [66] as a subroutine. This makes [66]'s overlapping idea still the state-of-the-art approach. However, even this $O(n^2)$ time algorithm may not be feasible in practice when n is very large. Thus there is a need for developing a faster deterministic algorithm for solving the CPS problem.

The rest of the chapter is organized as follows. In Section 3.2 we formalize the problem and review the $O(n^2)$ algorithm of [66]. In Section 3.3 we present our deterministic algorithm JUMP in detail. Following the description of JUMP, we provide some intuitive explanation and basic analysis of the algorithm's performance gains. Next in Section 3.5 we conduct extensive experiments and compare JUMP with the existing approaches on real datasets. Some concluding remarks are given in Section 6.

3.2 Problem Formulation and an $O(n^2)$ Time Algorithm

Let $A = a_1, a_2, \ldots, a_n$ be any given sequence of real numbers and let l be the dimension, i.e., the length of the subsequences we are looking for. The problem of identifying the closest pair of subsequences in A, is to find two **non-overlapping** subsequences $A_1 = (a_j, a_j + 1, \ldots, a_{j+l-1})$ and $A_2 = (a_{i+j-1}, a_{i+j}, \ldots, a_{i+j+l-2})$ such that $i > l$, and the distance between A_1 and A_2 is the least from among all such pairs. We use **Euclidean distance** as the similarity metric in the chapter, but note that our algorithm could be extended to other distance measures as well.

To illustrate the existing $O(n^2)$ algorithm, notice that the original problem can be decomposed to $(n - l + 1)$ subproblems. Let these subproblems be referred to as $\mathcal{P}_i$, for $l < i \leq (n - l + 1)$. The subproblem $\mathcal{P}_i$ is the following: Compute the distance between the following pairs of subsequences of length l: $((a_j, a_{j+1}, \ldots, a_{j+l-1}), (a_{i+j-1}, a_{i+j}, \ldots, a_{i+j+l-2}))$, for $1 \leq j \leq (n - l + 1)$. Note that in these distance calculations, we can ignore any pair if elements $a_{n'}$ (for $n' > n$) appear in any of the two subsequences. Let the distance between the pair $((a_j, a_{j+1}, \ldots, a_{j+l-1}), (a_{i+j-1}, a_{i+j}, \ldots, a_{i+j+l-2}))$ be d_j^i, for $1 \leq j \leq (n - l + 1)$.

The existing $O(n^2)$ algorithm used in [66] takes advantage of the overlapping part of consecutive pairs in the following manner: The squared Euclidean distance is computed as

$$(d_j^i)^2 = (a_j - a_{i+j})^2 + \ldots + (a_{j+l-1} - a_{i+j+l-1)})^2 \tag{3.1}$$

Here consider $(d_j^i)^2$ as a block that is a summation of l elements, and we refer to these $(a_i - a_j)^2$ values as **elements** inside a **block** $((d_j^i)^2)$ throughout the rest of this chapter.

The next pair's squared distance is

$$\begin{aligned}(d^i_{j+1})^2 &= (a_{j+1} - a_{i+j+1})^2 + \ldots + (a_{j+l} - a_{i+j+l})^2 \\ &= (d^i_j)^2 - (a_j - a_{i+j})^2 + (a_{j+l} - a_{i+j+l})^2.\end{aligned} \tag{3.2}$$

$(d^i_1)^2$ can be computed using l multiplications. Also, $(d^i_2)^2$ can be obtained from $(d^i_1)^2$ in an additional $O(1)$ time. Likewise, $(d^i_3)^2$ can be obtained from $(d^i_2)^2$ in an additional $O(1)$ time; and so on. Thus the problem $\mathcal{P}_i$ can be solved sequentially in a total of $O(n)$ time (for any specific value of i, $1 \leq i \leq (n - l + 1)$). As a result, the CPS problem can be solved exactly in $O(n^2)$ time. We refer to this algorithm in the rest of this chapter as **N2Alg**. Compared with the straightforward brute-force algorithm that computes all pair-wise distances using a total of $O(n^2 l)$ time, N2Alg's biggest advantage is that its run time is independent of l, which means a significant advantage for large l values, e.g., $l \geq 10^2$.

3.3 Proposed JUMP Algorithm

JUMP algorithm is essentially an enhanced version of N2Alg, in the sense that for each subproblem $\mathcal{P}_i$, JUMP would try to eliminate some unnecessary multiplication computations and hence reduce the total running time.

JUMP needs to solve each $\mathcal{P}_i$ individually and as a result, the outer loop for the n subproblems are the same as in N2Alg. Inside each loop of solving $\mathcal{P}_i$, JUMP has two phases: Init and Jump. A variable b keeps the best squared distance encountered so far. This variable is updated in each computation if a shorter squared distance is found.

Consider the subproblem $\mathcal{P}_i$. In the Init phase, JUMP first computes $(d_1^i)^2$ exactly as in N2Alg. Let a pointer p_1 point to the beginning position a_1 and another pointer p_2 point to a_{l+1}. Next we remove the first element (which is $(a_1 - a_{1+i})^2$) from $(d_1^i)^2$ and obtain a temporary value

$$tmp = (d_1^i)^2 - (a_1 - a_{1+i})^2 \tag{3.3}$$

Also, we move the pointer p_1 right to position a_2. Now we compare this tmp against the best-so-far b, and if tmp is still larger than b, we can be sure that $(d_2^i)^2$ is larger than b because $(d_2^i)^2$ contains tmp plus an additional element $(a_{1+l} - ai + 1 + l)^2$ which is definitely ≥ 0. So here we do not need to explicitly compute $(a_{1+l} - a_{i+1+l})^2$ at this moment.

We continue our elimination from the beginning, which is where p_1 points to, until $tmp \leq b$. Note that we need to move p_1 right after each elimination, to make sure it always points to the beginning of our tmp block.

Without loss of generality, assume that when p_1 points to $a_k, k \leq l$, tmp starts to become $\leq b$. From now on we need to add back elements because now tmp is only a partial squared distance and we need to fill it to l elements. The trick here is that we do not start filling from where p_2 points to (recall p_2 points to a_{l+1}, the end of initial block for $(d_1^i)^2$), but rather start filling from position $p_1 + l - 1$, which is a_{k+l-1}. This is because we want to know what is the value of $(d_k^i)^2$, and the last element of $(d_k^i)^2$ is $(a_{k+l-1} - a_{i+k+l-1})^2$:

$$tmp \leftarrow tmp + (a_{k+l-1} - a_{i+k+l-1})^2 \tag{3.4}$$

We employ two other pointers q_1 and q_2: q_1 points to a_{k+l-1} and q_2 points to a_{k+l}. If tmp is still $\leq b$, we continue our adding back

$$tmp \leftarrow tmp + (a_{k+l-2} - a_{i+k+l-2})^2 \tag{3.5}$$

and move q_1 left by one position. We keep doing this as long as the pointer q_1 moves left but has not met p_2, and tmp is still a partial squared distance.

Now we have obtained two blocks identified by p_1 (staring of the first block), p_2 (end position of the first block + 1) and q_1 (starting of the second block), q_2 (end position of the second block + 1). Next phase is Jump.

These two blocks are updated in the Jump phase, and the updating strategy is:

- If $tmp \geq b$, always remove elements from the beginning of the first block, and move p_1 one position right.
- If $tmp \leq b$ and $q_2 < p_1 + l$, add elements at position $q_2 - 1$ in the second block, move q_2 one position right.
- If $tmp \leq b$ and $q_2 = p_1 + l$, add back elements from the beginning of the second block, and move q_1 one position left.

Using this strategy, there are several conditions that would happen in the Jump phase:

- Case 1: Switching between removing and adding, keep moving p_1 and q_1.
- Case 2: Keep adding back until q_1 meets p_2 but tmp is still $\leq b$. Then we fill up tmp and obtain the new best-so-far $b = tmp = (d_k^i)^2$. In this case we finish computing all the square operations on those elements and there are no computational savings.

- Case 3: Keep removing elements until p_1 meets p_2 but tmp is still $\geq b$. **At this moment, we do not have to perform any of the square operations of elements lying between p_2 and q_1, but directly make the pointer p_1 jump to the position q_1.**

In the last case, the entire first block is removed but tmp is still $\geq b$, which means that the second block is already $\geq b$. We can skip all the elements between p_2 and q_1 as they will contribute more on the squared distance, and all the $(d_j^i)^2, p_2 \leq j < q_1$ must contain the whole second block, which is for sure $\geq b$.

This is how the savings in computation happen and we call the algorithm JUMP because of the jumping of the pointer during the algorithm. In summary, one case is that JUMP computes all the elements and updates the best-so-far b, the other case is the algorithm jumps some unnecessary computations as the existing partial sum is already $\geq b$. The benefit comes from the second case. The empirical results show that the jumping case occurs quite a lot and can result in significant run time reductions. The algorithm is formally described in Algorithm 3.

In Algorithm 3, line 2 represents the loop over all the subproblems $\mathcal{P}_i$; lines 6 to 11 are to build the initial two blocks; starting from line 12, the jumping scheme is implemented and the if-else clauses cover all the cases discussed above; lines 10 and 13 are boundary checks; line 18 is the place that skipping occurs, which is responsible for time savings in this algorithm. For a better illustration, the idea of the pointers is illustrated in Figure 3.1.

Algorithm 3 JUMP

Input: Sequence $A = a_1, a_2, \ldots, a_n$, subsequence length l

Output: A pair of subsequences that has the least Euclidean distance

Set best-so-far $b = \text{INF}$
for $i = l + 1$ to n **do**
  Compute distance $(d_1^i)^2$
  Set $p_1 = 1, p_2 = l + 1$
  $tmp = (d_1^i)^2 - (a_{p_1-1} - a_{p_1+i-1})^2$
  while $tmp \geq b$ and $p_1 \leq p_2$ **do**
    $tmp = tmp - (a_{p_1} - a_{p_1+i})^2$ and $p_1 = p_1 + 1$
  end while
  Set $q_1 = p_1 + l - 1, q_2 = q_1 + 1$ if $p_1 + i \leq n - l$
  Otherwise Break
  $tmp = tmp + (a_{q_1} - a_{q_1+i})^2$
  while 1 **do**
    if $p_1 + i > n - l$ **then**
      Break
    else if $tmp \geq b$ and $p_1 < p_2$ **then**
      $tmp = tmp - (a_{p_1} - a_{p_1+i})^2$ then $p_1 = p_1 + 1$
    else if $tmp \geq b$ and $p_1 = p_2$ **then**
      Set $p_1 = q_1, p_2 = q_2, q_1 = p_1 + l - 1, q_2 = q_1 + 1$
      $tmp = tmp + (a_{q_1} - a_{q_1+i})^2$
    else if $tmp < b$ and $q_2 \leq p_1 + l - 1$ **then**
      $tmp = tmp + (a_{q_2} - a_{q_2+i})^2$ then $q_2 = q_2 + 1$
    else if $tmp < b$ and $q_1 > p_2$ **then**
      $q_1 = q_1 - 1$ then $tmp = tmp + (a_{q_1} - a_{q_1+i})^2$
    end if
    if $q_1 = p_2$ and $q_2 = p_1 + l$ **then**
      if $tmp < b$ **then**
        $b = tmp$ and $\text{loc}_1 = p_1, \text{loc}_2 = p_1 + i$
      end if
      $p_1 = p_1 + 1, q_1 = p_1 + l - 1, p_2 = q_1, q_2 = q_1 + 1$
      $tmp = tmp - (a_{p_1-1} - a_{p_1-1+i})^2 + (a_{q_1} - a_{q_1+i})^2$
    end if
  end while
end for
return The least Euclidean distance as $\sqrt{b}$ and the corresponding subsequence starting positions as $(\text{loc}_1, \text{loc}_2)$

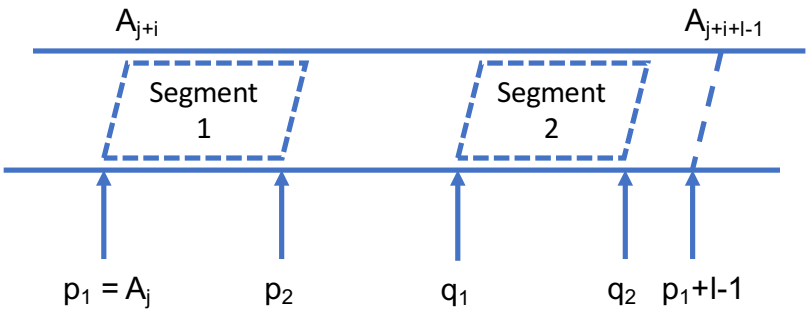


FIGURE 3.1: Illustration of JUMP: p_1, p_2 are beginning and ending pointers for the first block, q_1, q_2 are for the second segment. Vertical lines represent alignments (elements e in the distance block $(d_j^i)^2$, e.g., $e_j = (a_{j+i} - a_j)^2$).

3.4 A Brief Analysis of JUMP

In this section we take a deeper look into the algorithm and provide some intuitive explanation on the performance behavior of the JUMP algorithm. In particular, we are interested in seeing how many multiplications (square operations) could be saved during the process compared with N2Alg. Note that this is not a formal proof but an explanatory analysis for a better understanding.

3.4.1 The Best-so-far b:

The key idea behind JUMP is to skip unnecessary multiplication computations. In a block of consecutive l elements, we can jump over some of the elements only if the value of tmp is larger than best-so-far, So the first thing of interest is how best-so-far b value changes during the algorithm. For a simple illustration, in Figure 3.2, we plot b during running of N2Alg (JUMP has the same framework and hence has a similar b) algorithm on several numerical sequences. The numbers in two of the sequences are uniformly distributed in $[0, 1]$, while those in the other two are normally distributed

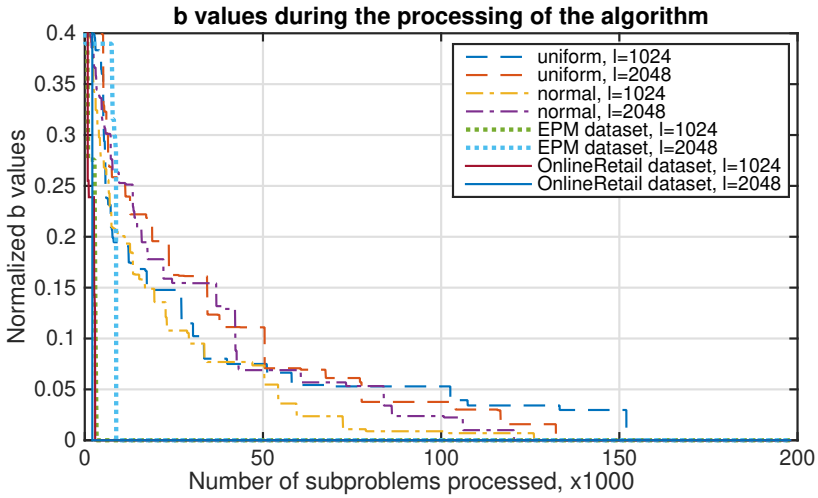


FIGURE 3.2: b values change during processing of the algorithm

as $N(0,1)$. We also plot b for two real datasets that are used in Section 3.5. The b values are taken as averages over 10 runs. All the values are normalized into $[0,1]$.

Intuitively we know that if b drops fast at the beginning, then in the later processing JUMP would have a higher chance to skip elements. As shown in Figure 3.2, the initial value of b can be seen as the expectation of pairwise distances. Then in the process of the algorithm, b drops significantly, especially in the real datasets that have a periodic behavior (e.g., online retail data). This provides JUMP the potential of skipping a large fraction of the calculations.

3.4.2 Estimate the Number of Skips:

Given b, we can estimate how many elements we need in each block of length l, such that the sum of these elements could be $\geq b$. To simplify the estimation, assume that each element $e_j = (a_j - a_{j+i})^2$ is an independent random variable and e_j is bounded $\in [z_1, z_2]$. In a block of length l, denote the partial-sum of k elements as

$((d_j^i)_k)^2 = \sum_{p=0}^{k-1} e_{j+p}$. Let $\mathbf{E}[e_j] = \mu$, thus $\mathbf{E}[((d_j^i)_k)^2] = k\mu$. Using the

Hoeffding bound: Let $X_1, X_2, \ldots, X_k$ be independent random variables (no distribution assumption) such that $z_1 \leq X_i \leq z_2$ for all i. Let $X = \sum_1^k X_i$, then for any $t > 0$, $\mathbf{P}[X - \mathbf{E}[X] \leq -t] \leq e^{-\frac{2t^2}{k(z_2-z_1)^2}}$.

If we set $k\mu - t = b$, this leads to

$$\mathbf{P}[(d_j^i)_k^2 \leq b] = \mathbf{P}[\text{sum of } k \text{ elements} \leq b] \leq e^{-\frac{2(k\mu-b)^2}{k(z_2-z_1)^2}} \tag{3.6}$$

The above inequality shows the probability for the sum of k elements to be less than b. For appropriate values of b, this probability can be low. In other words, if b takes on an appropriate value, it is very likely for the sum of the k elements to be $> b$. This in turn will mean that the number of skips can be as large as $l - k$. In fact this happens quite often in practice as revealed in our experimental data described in Section 3.5. Let

$$\mathbf{P_k}(b) = 1 - \mathbf{P}[(d_j^i)_k^2 \leq b] \geq e^{-\frac{2(k\mu-b)^2}{k(z_2-z_1)^2}}. \tag{3.7}$$

$\mathbf{P_k}(b)$ is an upper bound of the probability that JUMP can skip $(l - k)$ elements. Since k ranges from 1 to l, taking the expectation would give us how many elements can be skipped at most in a block of length l:

$$\mathbf{E}[\#\text{ of skipped elements}] \leq \sum_{k=1}^{l} (l-k)\mathbf{P_k}(b). \tag{3.8}$$

Since there are $\lfloor \frac{n-l-i+1}{l} \rfloor$ blocks of length l in subproblem $\mathcal{P}_i$, and there are $(n-l+$

1) subproblems, the total number of block is $M \approx \sum_{i=1}^{n-l+1} \lfloor \frac{n-i}{l} \rfloor \approx \frac{n^2}{2l}$. The maximum total number of skips is a sum of maximum skips over all the blocks:

$$\mathbf{E}[\text{total \# of skips}] \leq \sum_{m=1}^{n^2/2l} \sum_{k=1}^{l} (l-k)\mathbf{P_k}(b(m)). \tag{3.9}$$

In the above equation $b(m)$ stands for b to indicate that b could be a function of m, since b changes from block to block during the processing. Also, in the above equation, R.H.S. $\geq \sum_{m=1}^{n^2/2l} \sum_{k=1}^{l} (l-k) e^{-2\frac{(k\mu - b(m))^2}{k(z_2-z_1)^2}}$.

3.4.3 Some Discussion:

Here we provide some discussion on the elementary analysis above.

- Although the elementary analysis only gives an upper bound on the total number of skips, it could offer a basic idea on how JUMP performs.

- For a more formal and accurate analysis, a better estimation of b is needed. Brownian Motion or Markov process could be suitable models to obtain b values.

- The assumption that each element $e = (a_i - a_j)^2$ satisfies $z_1 \leq e \leq z_2$, is usually valid in practice. For example, if the subsequence $A_i = a_i, a_{i+1}, \ldots, a_{i+l-1}$ is a point in a normalized hypercube $[0,1]^l$, then $z_1 = 0$ and $z_2 = 1$ are tight bounds. Even if there are some outliers, we can ignore them and still use z_1, z_2 to bound the concentration of e.

- Intuitively, if l is large, it becomes harder for a small fraction of elements to have a sum of more than b. In other words, JUMP should perform better for smaller l values. On the contratry, as the total sequence length n goes up, from

Equation 3.9 we can easily see that the maximum number of skips could be more and hence JUMP should have larger gains. These relationships are also verified in the following experiments.

3.5 An Experimental Evaluation of JUMP

In this section, we have performed extensive experiments on existing standard benchmark sequence datasets to evaluate our JUMP algorithm. The test server has an Intel Xeon CPU E5-2667 v3 @ 3.2 GHz. We have implemented the algorithms in C/C++, and used gcc as the default compiler. The source code can be found at https://github.com/TideDancer/JUMP.

3.5.1 Real Data Evaluation:

In order to see the performance on standard benchmark datasets, we have randomly chosen several datasets from UCI Machine Learning Repository [45] and used them to evaluate our proposed JUMP algorithm. These datasets cover different sizes and different types that ensure an unbiased evaluation. All these datasets are numerical sequences with either integer or real numbers, and some of them are time series datasets. Next we provide detailed experimental description and analysis for small datasets and large datasets, respectively.

Educational Process Mining (EPM): A Learning Analytics Data Set: First we start with a small dataset with round 200k numbers in the sequence. This data set contains some students' time series of activities during sessions of laboratory course of digital electronics. We use the mouse movement as the sequence data to test

TABLE 3.1: Performance evaluation on EPM dataset

$n = 2 \times 10^5$	N2Alg	JUMP	Skip Fraction
$l = 512$	41.78	0.98	0.981
$l = 1024$	42.24	9.10	0.895
$l = 2048$	42.18	21.04	0.788
$l = 4096$	39.58	22.35	0.712

TABLE 3.2: Performance evaluation on Online Retail data

$n = 5 \times 10^5$	N2Alg	JUMP	Skip Fraction
$l = 1024$	263.54	137.81	0.790
$l = 2048$	263.24	105.56	0.802
$l = 4096$	261.51	71.72	0.814
$l = 10 \times 10^3$	257.35	108.47	0.687

our algorithms, and the sequence only has integer numbers.. Note that one possible application for our algorithms is in finding the two most similar behavior patterns (with the smallest Euclidean distance) of the students during the whole session, where the length of the pattern is l. We compare run times (in seconds) of the JUMP and N2Alg in different settings and show the results in Table 3.1.

Online Retail Data Set: This is a small size dataset with around 500k entries. The dataset contains transactions occurring during a certain period of time for a UK-based online retail. We use the prices data to form the entire sequence. The run times (in seconds) are shown in Table 3.2.

Individual household electric power consumption Data Set [45]: This is a moderate size dataset with around 2 million entries. It is a measurement of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. Four sets of numbers (active power, reactive power, intensity, voltage) are used to form four sequences and both the algorithms have been run on these sequences. We set $n = 1 \times 10^6$ to use the first 1 million entries, and set

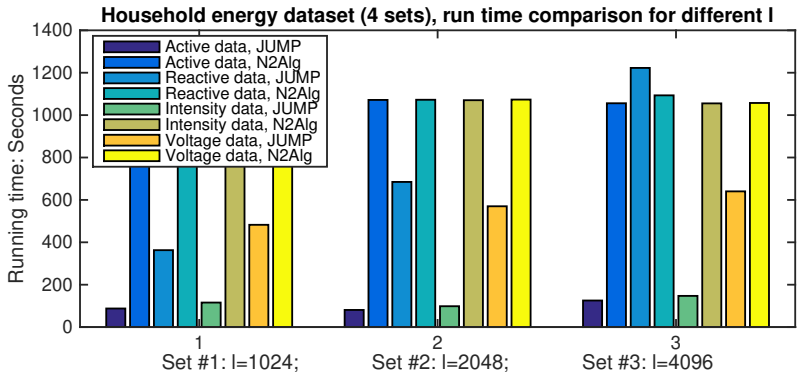


FIGURE 3.3: Run time for Household energy dataset, $n = 1 \times 10^6$ with different l values

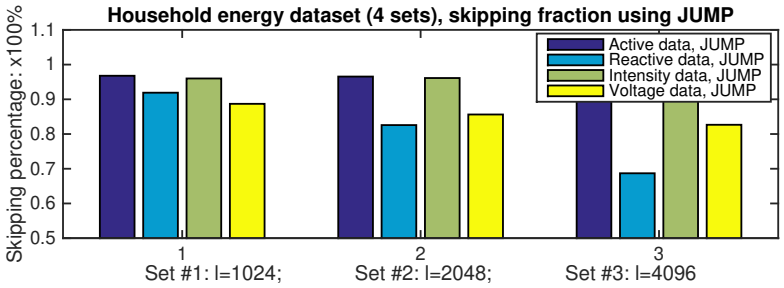


FIGURE 3.4: Skipping fraction for Household energy dataset, $n = 1 \times 10^6$ with different l values

$l = 1024, 2048$ and $20k$. The run time comparison is given in Figure 3.3, while the skipping percentage is provided in Figure 3.4.

From Figure 3.3 and Figure 3.4 we see that JUMP is generally much faster than N2Alg. Also, JUMP's performance depends on how much percentage it can skip during the process, which depends on the nature of the dataset. As we can see, the Reactive dataset gives worst performance for JUMP.

To see how the performance of both the algorithms varies as the dimension increases, we have set $l = 1k$ to $30k$ at an interval of $5k$, where n is 1 million, using the active power data. The result is shown in Figure 3.5's first plot. We note that for all the different l values, JUMP could outperform N2Alg. In the second plot of Figure 3.5, the jumping fraction is illustrated, where we see that as l increases, the

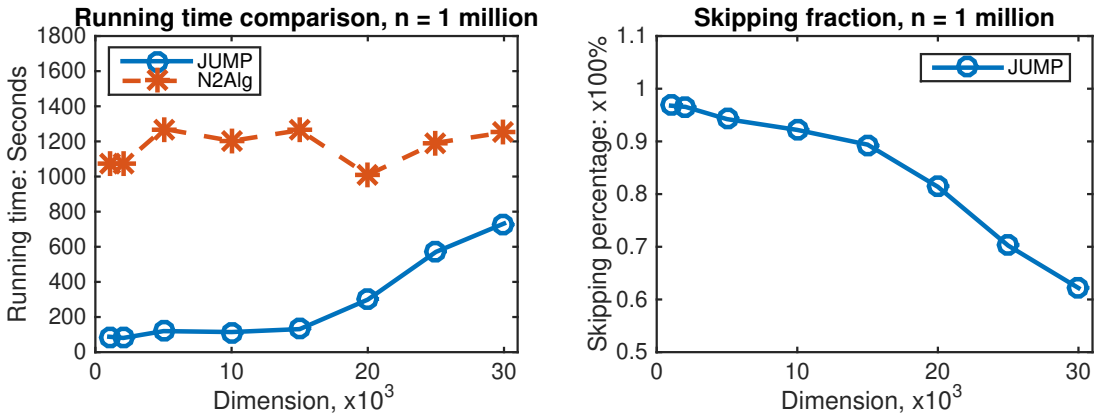


FIGURE 3.5: Household-Active dataset, $n = 1 \times 10^6$, l ranges from $1k$ to $30k$

percentage of multiplication operations that JUMP could skip decreases, which has an impact on the run time as well. However, even for large dimensions up to 30k (note that the entire sequence length is only 1 million), the skipping fraction is still satisfactory and the run time is still much less than that of N2Alg.

Heterogeneity Activity Recognition Data Set: Here we evaluate both the algorithms on large (large sequence lengths and large dimensions) data. This dataset contains cellphone accelerometer and gyroscope data of human activity. As the cellphone sensors' sampling rate is high, the dataset is large having around 1×10^7 entries. We use the x, y and z axes recorded from both the accelerometer and the gyroscope to evaluate JUMP and compare it with N2Alg.

In these experiments, we first show how these two algorithms perform as n increases. n is set from 0.5×10^6 to 5×10^6, and the first n entries from the Accelerometer-X-axis dataset are used. We show the results for both the large dimension case ($l = 50k$) and a moderate dimension case ($l = 5k$) as they are more challenging.

Figure 3.6 (a Semilog-Y plot) shows how the performance differs as n increases. Since N2Alg is an $O(n^2)$ time algorithm, the performance decays quadratically as

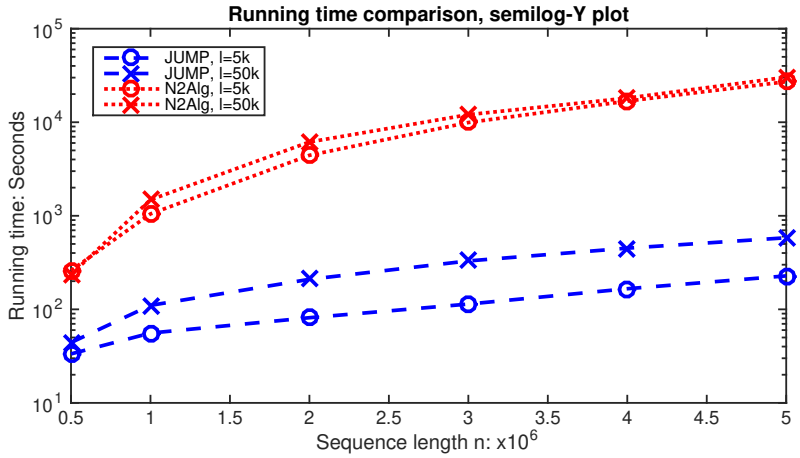


FIGURE 3.6: Running time for different sequence length n on Accelerometer-X data, dimension $l = 50k, 5k$

expected. In contrast, JUMP has a near linear run time, thus has a much better performance when n is large. Also, the skipping fraction and speedup of JUMP algorithm are shown in Figure 3.7, from which we note that because the skipping percentage grows as n increases, JUMP's run time grows slower than that of N2Alg such that the speedup is increasing.

From both the figures we observe that JUMP's performance is better in lower dimensions (e.g., the $l = 5k$ case), which is consistent with the results from the other experiments. However, even in super large dimensions like $l = 50k$, JUMP is still significantly better than N2Alg.

Next is a full evaluation using different settings. We choose n to be 2×10^6 and 4×10^6, and set l to be $1k$ to $100k$. We compare the run times of JUMP and N2Alg on our test server and first provide the results for $n = 2 \times 10^6, l = 1024, 2048$ in Table 3.3. Clearly, nearly 100x speedup could be achieved in this setting on Accelerometer-X axis data, while the speedup is nearly 50x on the Gyroscope-X axis data.

For $n = 4 \times 10^6$ and dimensions $l = 5k, 10k, 25k, 50k, 75k, 100k$, the performance

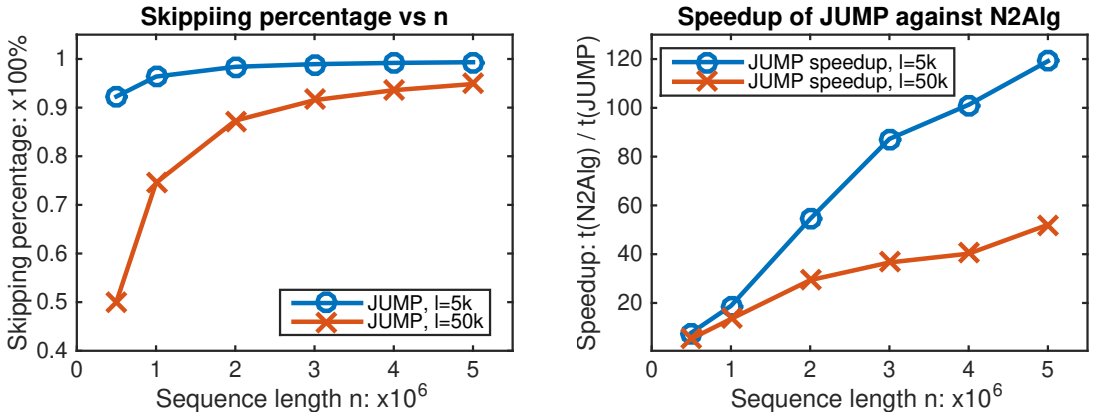


FIGURE 3.7: JUMP's speedup and skipping percentage for different sequence length n on Accelerometer-X data, dimension $l = 50k, 5k$

TABLE 3.3: Comparison on Activity data, $n = 2m$

		Accelerometer-X		Gyroscope-X	
l	Axis	N2Alg	JUMP	N2Alg	JUMP
1024	x	4368.94	65.16	4373.08	117.58
2048	x	4370.6	71.51	4363.93	108.01
1024	y	4382.45	73.05	4365.65	101.36
2048	y	4393.52	70.27	4365.17	106.48
1024	z	4373.3	63.52	4378.66	110.09
2048	z	4364.74	60.27	4377.2	109.71

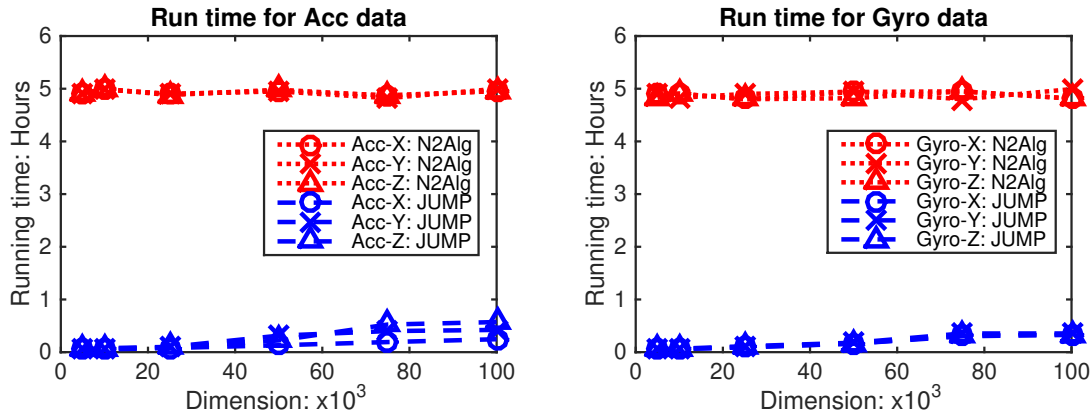


FIGURE 3.8: Run time comparison on accelerometer and gyroscope data, $n = 4 \times 10^6$, l ranges from $5k$ to $100k$

comparison is shown in Figure 3.8 and Figure 3.9. The first plot illustrates the comparison on the accelerometer dataset; the second plot shows the result for the gyroscope dataset; and the third plot describes the speedup of JUMP against N2Alg.

When the dimension is large such as 10×10^3, the speedup of JUMP over N2Alg is 100. Even in extreme cases like $l = 100 \times 10^3$, 20x speedup is still expected.

3.5.2 Summary of experiments:

From the above experiments on standard benchmark datasets, we make the following observations:

- As the total sequence length n increases, JUMP's skipping fraction increases resulting in a huge speedup.
- If the subsequence length l is extremely large (e.g., $l > 100k$ when $n = 4$ million), JUMP's speedup decreases significantly.
- Empirical results show that if the skipping fraction drops to around 50%, JUMP is no longer faster than the existing approach N2Alg.

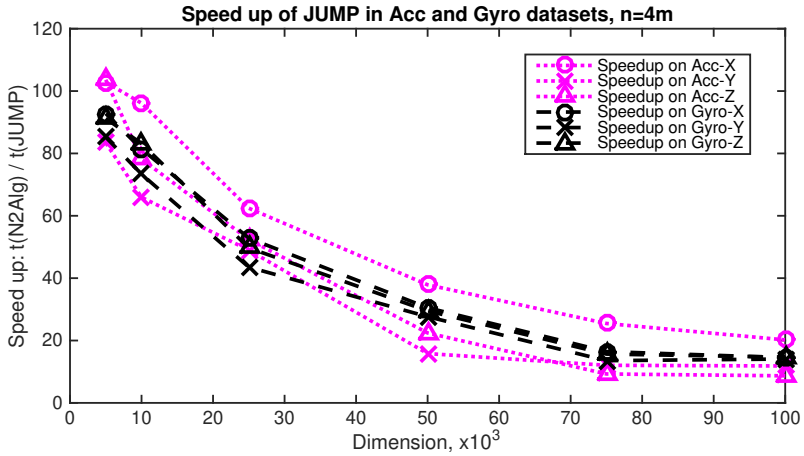


FIGURE 3.9: Speedup comparison on accelerometer and gyroscope data, $n = 4 \times 10^6$, l ranges from $5k$ to $100k$

- On large datasets, with a moderate dimension l, JUMP could achieve 100x speedup (the skipping fraction is more than 95%). Even with large l in our settings, 20x speedup is still achievable when 85% or more could be skipped.

Based on the above observations we could clearly see how n and l affect JUMP's skipping fraction and further affect the run time of JUMP. Due to the more sophisticated logic in JUMP, the overhead in implementation could offset the advantage by jumping unnecessary multiplications. However, even though modern computers are equipped with fast pipeline multiplication units that take $O(1)$ time for doing the multiplication or the square operation, the huge savings that JUMP offers could result in a speedup of up to 100x compared with the state-of-the-art algorithms. Empirical results show that only when the dimension is extremely large, JUMP would lose its advantage due to the decrease in skipping fractions. So JUMP offers a much better performance on an average in these benchmarks, and furthermore, it could be very helpful in embedded systems that do not have advanced multipliers.

3.6 Conclusions and Future Work

In this chapter we study the Closest Pair of Subsequences problem and propose a deterministic algorithm that solves this problem very efficiently. By looking into the overlapping parts of consecutive subsequences, JUMP always maintains a lower bound during the processing and skips a significant amount of unnecessary computations. A brief explanatory analysis of the JUMP algorithm is also provided to illustrate the intuition behind the algorithm. Extensive experiments are carried out to evaluate the performance of JUMP on standard benchmarks. Different types of test data are used in the experiments to fully evaluate JUMP in every aspect. In the end of Section 3.5, we summarize our observations and provide an analysis on the experimental results.

In our future work, we will perform additional experiments on more benchmarks to further test JUMP's performance. A deeper study on the relationship between sequence total length n, subsequence dimension l, and the skipping fraction of JUMP could be conducted. The idea of jumping unnecessary operations in sequence mining could be also applied to other sequence processing and pattern finding applications.

Chapter 4

The Closest Pair Pattern Mining in Dynamic Time Warping

4.1 Introduction

In this chapter, we switch our distance measure, and bring the machine learning perspective to address the pattern mining problem. Instead of a conventional predefined pattern search, we illustrate a neural network based framework to perform a learnable target identification and show how it serves different ending tasks.

In many data mining and machine learning problems, a proper metric of similarity or distance could play a significant role in the model performance. Minkowski distance, defined as $\text{dist}(x, y) = (\sum_{k=1}^{d} |x_k - y_k|^p)^{1/p}$ for input $x, y \in \mathcal{R}^d$, is one of the most popular metrics. In particular, when $p = 1$, it is called Manhattan distance; when $p = 2$, it is the Euclidean distance. Another popular measure, known as Mahalanobis distance, can be viewed as the distorted Euclidean distance. It is defined as

$\text{dist}(x, y) = ((x - y)^T \Sigma^{-1}(x - y))^{1/2}$, where $\Sigma \in \mathcal{R}^{d \times d}$ is the covariance matrix. With geometry in mind, these distance (or similarity) measures, are straightforward and easy to represent.

However, in the domain of sequence data analysis, both Minkowski and Mahalanobis distances fail to reveal the true similarity between two targets. Dynamic Time Warping (DTW) [4] has been proposed as an attractive alternative. The most significant advantage of DTW is its invariance against signal warping (shifting and scaling in the time axis, or Doppler effect). Therefore, DTW has become one of the most preferable measures in pattern matching tasks. For instance, two different sampling frequencies could generate two pieces of signals, while one is just a compressed version of the other. In this case, it will be very dissimilar and deviant from the truth to use the point-wise Euclidean distance. On the contrary, DTW would capture such scaling nicely and output a very small distance between them. DTW not only outputs the distance value, but also reveals how two sequences are aligned against each other. Sometimes, the alignment could be more interesting. Furthermore, DTW could be leveraged as a feature extracting tool, and hence it becomes much more useful than a similarity measure itself. For example, predefined patterns can be identified in the data via DTW computing. Subsequently these patterns could be used to classify the temporal data into categories, e.g., [36]. Some interesting applications can be found in, e.g., [26, 55].

The standard algorithm for computing Dynamic Time Warping involves a Dynamic Programming (DP) process. With the help of $O(n^2)$ space, a cost matrix C would be built sequentially, where

$$C_{i,j} = ||x_i - y_j|| + \min\{C_{i-1,j}, C_{i,j-1}, C_{i-1,j-1}\} \tag{4.1}$$

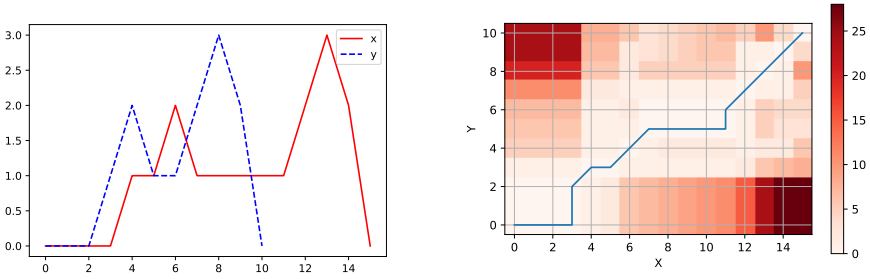


FIGURE 4.1: DTW path that aligns x and y

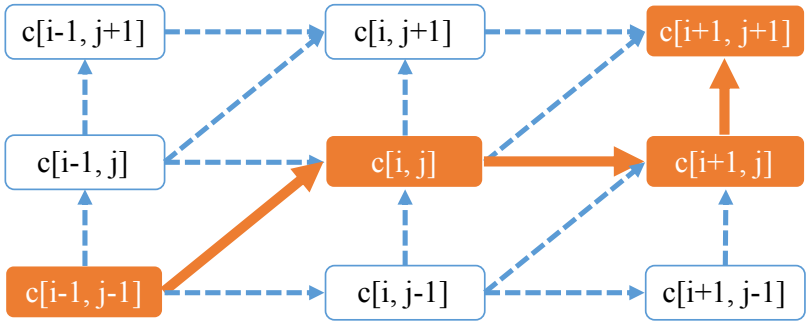


FIGURE 4.2: The path is fixed after DP

Here $||x_i - y_j||$ denotes the norm of $(x_i - y_j)$, e.g., p-norm, $p = 1, 2$ or ∞. After performing the DP, we can trace back and identify the warping path from the cost matrix. This is illustrated in Figure 4.1, where two sequences of different lengths are aligned. There are speedup techniques to reduce DTW's time complexity, e.g., [56], which is beyond the scope of this chapter. In general, a standard DP requires $O(n^2)$ time.

Although DTW is already one of the most important similarity measures and feature extracting tools in temporal data mining, it has not contributed much to the recent deep learning field. As we know, a powerful feature extractor is the key to the success of an artificial neural network (ANN). The best example could be the CNNs that utilize convolutional kernels to capture local and global features [39]. Unlike the

convolution, DTW has the non-linear transformation property (warping), providing a summary of the target against Doppler effects. This makes DTW a good candidate as a feature extractor in general ANNs. With this motivation, we propose DTWNet, a neural network with learnable DTW kernels.

Key Contributions: We apply the learnable DTW kernels in neural networks to represent Doppler invariance in the data. To learn the DTW kernel, a stochastic backpropogation method based on the warping path is proposed, to compute the gradient of a DP process. A convergence analysis of our backpropogation method is offered. To the best of the authors' knowledge, for the first time, DTW loss function is theoretically analyzed. A differentiable streaming DTW learning is also proposed to overcome the problem of missing local features, caused by global alignment of the standard DTW. Empirical study shows the effectiveness of the proposed backpropogation and the success of capturing features using DTW kernels. We also demonstrate a data decomposition application.

4.2 Related Work

4.2.1 Introduction of Dynamic Time Warping

Dynamic Time Warping is a very popular tool in temporal data mining. For instance, DTW is invariant of Doppler effects thus it is very useful in acoustic data analysis [55]. Another example is that biological signals such as ECG or EEG, could use DTW to characterize potential diseases [79]. DTW is also a powerful feature extractor in conjunction with predefined patterns (features), in the time series classification problem [36]. Using Hamming distance, the DTW alignment in this setting is called

the Edit distance and also well studied [27].

Due to the Dynamic Programming involved in DTW computation, the complexity of DTW can be high. More critically, DP is a sequential process which makes DTW not parallelizable. To speedup the computation, some famous lower bounds based techniques [37, 42, 72] have been proposed. There are also attempts on parallelization of DP [80] or GPU acceleration [74].

Two dimensional DTW has also drawn research interests. In [40], the author showed that the DTW could be extended to the 2-D case for image matching. Note that this is different from another technique called **multi-variate DTW** [77, 51, 73], sometimes also referred to as multi-dimensional DTW. In multi-variate DTW, the input is a set of 1-D sequences, e.g., of dimension $k \times n$ where n is the sequence length. However, in 2-D or k-D DTW, the input is no longer a stack of 1-D sequences but images (n^2) or higher dimensional volumes (n^k). As a result, the cost of computing 2-D DTW can be as high as $O(n^6)$ and thus making it not applicable for large datasets.

4.2.2 SPRING Algorithm, the Streaming Version of DTW

To process the streaming data under DTW measure, [69] proposed a modified version of DTW computation called SPRING. The original DTW aims to find the best alignment between two input sequences, and the alignment is from the beginning of both sequences to the end. On the contrary, the streaming version tries to identify all the subsequences from a given sequence, that are close to a given pattern under the DTW measure. The naive approach computes DTW between all possible subsequences and the pattern. Let the input sequence and the pattern be of lengths n and l, respectively. The naive method takes $(nl + (n-1)l + \ldots) = O(n^2 l)$ time. However,

SPRING only takes $O(nl)$ time, which is consistent with the standard DTW.

SPRING modifies the original DTW computation with two key factors. First, it prepends one wild-card to the pattern. When matching the pattern with the input, since the wild-card can represent any value, the start of the pattern could match any position in the input sequence at no cost. The second modification is that SPRING makes use of an auxiliary matrix to store the source of each entry in the original dynamic programming matrix. This source matrix will keep records of each candidate path and hence we can trace back from the end. Interested readers could refer to [69] for more details.

4.2.3 DTW as a Loss Function

Recently, in order to apply the DTW distance for optimization problems, the differentiability of DTW has been discussed in the literature. As we know, computing DTW is a sequential process in general. During the filling of the DP matrix, each step takes a min operation on the neighbors. Since the min operator is not continuous, the gradient or subgradient is not very well defined. The first attempt to use soft-min function to replace min is reported in [70]. In their paper, the authors provide the gradient of soft-min DTW, and perform shapelet learning to boost the performance of time series classification in limited test datasets. Using the same soft-min idea, in [17], the authors empirically show that applying DTW as a loss function leads to a better performance than conventional Euclidean distance loss, in a number of applications. Another very recent paper [13] also uses continuous relaxation of the min operator in DTW to solve video alignment and segmentation problems.

4.3 Proposed DTW Layer and its Backpropogation

In this chapter, we propose to use DTW layers in a deep neural network. A DTW layer consists of multiple DTW kernels that extract meaningful features from the input. Each DTW kernel generates a single channel by performing DTW computation between the kernel and the input sequences. For regular DTW, one distance value will be generated for each kernel. For the streaming DTW, multiple values would be output (details will be given in § 4.5). If using a sliding window, the DTW kernel would generate a sequence of distances, just as a convolutional kernel. After the DTW layer, linear layers could be appended, to obtain classification or regression results. A complete example of DTWNet on a classification task is illustrated in Algorithm 4.

Algorithm 4 DTWNet training for a classification task. Network parameters are: number of DTW kernels N_{kernel}; kernels $x_i \in \mathcal{R}^l$; linear layers with weights w.

Input: Dataset $Y = \{(y_i, z_i) | y_i \in \mathcal{R}^n, z_i \in \mathcal{Z} = [1, N_{\text{class}}]\}$. The DTWNet dataflow can be denoted as $\mathcal{G}_{x,w} : \mathcal{R}^n \to \mathcal{Z}$.

Output: The trained DTWNet $\mathcal{G}_{x,w}$

1: Init w; For $i = 1$ to N_{kernel}: randomly init x_i; Set total # of iteration be T, stopping condition ϵ
2: **for** $t = 0$ to T **do**
3: Sample a mini-batch $(y, z) \in Y$. Compute DTWNet output: $\hat{z} \leftarrow \mathcal{G}_{x,w}(y)$
4: Record warping path $\mathcal{P}$ and obtain determined form $f_t(x, y)$, as in Equation 4.2
5: Let $\mathcal{L}_t \leftarrow \mathcal{L}_{CrossEntropy}(\hat{z}, z)$. Compute $\nabla_w \mathcal{L}_t$ through regular BP.
6: For $i = 1$ to N_{kernel}: compute $\nabla_{x_i} \mathcal{L}_t \leftarrow \nabla_{x_i} f_t(x_i, y) \frac{\partial \mathcal{L}_t}{\partial f_t}$ based on $\mathcal{P}$, as in Equation 4.3
7: SGD Update: let $w \leftarrow w - \alpha \nabla_w \mathcal{L}_t$ and for $i = 1$ to N_{kernel} do $x_i \leftarrow x_i - \beta \nabla_{x_i} \mathcal{L}_t$
8: If $\Delta \mathcal{L} = |\mathcal{L}_t - \mathcal{L}_{t-1}| < \epsilon$: return $\mathcal{G}_{x,w}$
9: **end for**

Gradient Calculation and Backpropogation

To achieve learning of the DTW kernels, we propose a novel gradient calculation and backpropogation (BP) approach. One simple but important observation is that: after performing DP and obtaining the warping path, the path itself is settled down for this iteration. If the input sequences and the kernel are of lengths n and l, respectively, the length of the warping path cannot be larger than $O(n+l)$. This means that the final DTW distance could be represented using $O(n+l)$ terms, and each term is $||y_i - x_j||$ where $i, j \in \mathcal{S}$, and $\mathcal{S}$ is the set containing the indices of elements along the warping path. For example, if we use 2-norm, the final squared DTW distance could be of the following form:

$$\text{dtw}^2(x, y) = f_t(x, y) = ||y_0 - x_0||_2^2 + ||y_1 - x_0||_2^2 + ||y_2 - x_1||_2^2 + \dots \tag{4.2}$$

This is illustrated in Figure 4.2, where the solid bold lines and the highlighted nodes represent the warping path after Dynamic Programming. Since the warping path is determined, other entries in the cost matrix no longer affect the DTW distance, thus the differentiation can be done only along the path. Since the DTW distance obtains its determined form, e.g., Equation 4.2, taking derivative with respect to either x or y becomes trivial, e.g.,

$$\nabla_x \text{dtw}^2(x, y) = \nabla_x f_t(x, y) = [2(y_0 + y_1 - 2x_0)\ ,\ 2(y_2 - x_1)\ ,\ \dots]^T \tag{4.3}$$

Since the min operator does not have a gradient, directly applying auto-diff will result in a very high variance. Soft-min could somewhat mitigate this problem, however, as shown above, since the final DTW distance is only dependent on the elements

along the warping path, differentiation on all the entries in the cost matrix becomes redundant. Other than this, additional attention needs to be paid to the temperature hyperparameter in the soft-min approach, which controls the trade-off between accuracy and numerical stability.

In contrast, taking derivative using the determined form along the warping path, we can avoid the computation redundancy. As the warping path length cannot exceed $O(n + l)$, the differentiation part only takes $O(n + l)$ time instead of $O(nl)$ as in the soft-min approaches. Note that there is still a variance which arises from the difference in DP's warping paths from iteration to iteration, so the BP can be viewed as a stochastic process.

Time Complexity: The computation of DTW loss requires building a Dynamic Programming matrix. The standard DP needs $O(nl)$ time. There are speeding-up/approximating techniques for DP such as banded constraint (limit the warping path within a band), which is beyond the scope of this chapter. The gradient is evaluated in $O(n+l)$ time as shown above. Although the DP part is not parallelizable in general, parallelization can still be achieved for independent evaluation for different kernels.

4.4 DTW Loss and Convergence

To simplify the analysis, we consider that for one input sequence $y \in \mathcal{R}^n$. The goal is to obtain a target kernel $x \in \mathcal{R}^l$ that has the best alignment with y, i.e., $\min_x \mathrm{dtw}^2(x, y)$. Without loss of generality, we assume $l \leq n$. The kernel x is randomly initialized and we perform learning through standard gradient descent. Define

the DTW distance function as $d = H_y(x)$, where $d \in \mathcal{R}$ is the DTW distance evaluated by performing the Dynamic Programming operator, i.e., $d = \mathrm{DP}(x, y)$.

Definition 1. *Since DP provides a deterministic warping path for arbitrary x, we define the space of all the functions of x representing all possible warping paths as*

$$\mathcal{F}_y = \{f_y(x) | f_y(x) = \sum_{i,j} I_{ij} ||(x_i - y_j)||_2^2\}$$

$$s.t. \quad i \in [0, l-1];\ j \in [0, n-1];\ I_{ij} \in \{0, 1\};\ n \leq |I| \leq n + l;$$

$$i, j \text{ satisfy temporal order constraints.}$$

Here the cardinality of I is within the range of n and $n + l$, because the warping path length can only be between n and $n + l$. The temporal order constraints make sure that the combination of i, j must be valid. For example, if x_i is aligned with y_j, then x_{i+1} cannot be aligned with y_{j-1}, otherwise the alignment will be against the DTW definition.

With Definition 1, when we perform Dynamic Programming at an arbitrary point x to evaluate $H_y(x)$, we know that it must be equal to some function sampled from the functional space $\mathcal{F}_y$, i.e., $H_y(x)|_{x=\hat{x}} = f_y^{(u)}(x)|_{x=\hat{x}}$, $f_y^{(u)} \in \mathcal{F}_y$. So we can approximate $H_y(x)$ as a collection of functions in $\mathcal{F}_y$, where each x could correspond to its own sample function. In the proposed backpropogation step we compute the gradient of $f_y^{(u)}(x)$ and perform the gradient descent using this gradient. The first question is whether $\nabla_x f_y^{(u)}(x)|_{x=\hat{x}} = \nabla_x H_y(x)|_{x=\hat{x}}$.

We notice the fact that $H_y(x)$ is not smooth in the space of x. More specifically,

there exist positions x such that

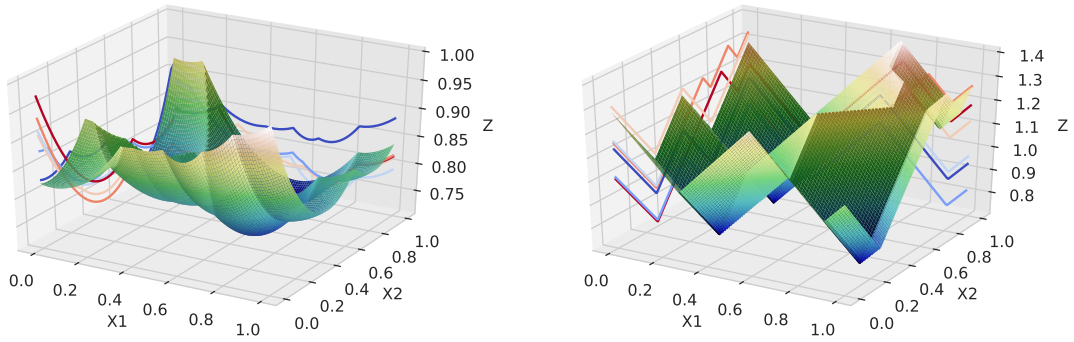
$$H_y(x) = \begin{cases} f_y^{(u)}(x)|_{x=x_+} \\ f_y^{(v)}(x)|_{x=x_-} \end{cases} \quad u \neq v; \quad f_y^{(u)}, f_y^{(v)} \in \mathcal{F}_y \tag{4.4}$$

where x_+ and x_- represent infinitesimal amounts of perturbation applied on x, in the opposite directions. However, note that the cardinality of $\mathcal{F}_y$ is finite. In fact, in the Dynamic Programming matrix, for any position, the warping path can only evolve in at most three directions, due to the temporal order constraints. In boundary positions, only one direction can the warping path evolve along. So we have:

Lemma 4.4.1. *Warping paths number $|\mathcal{F}_y| < 3^{n+l}$, where $(n+l)$ is the largest possible path length.*

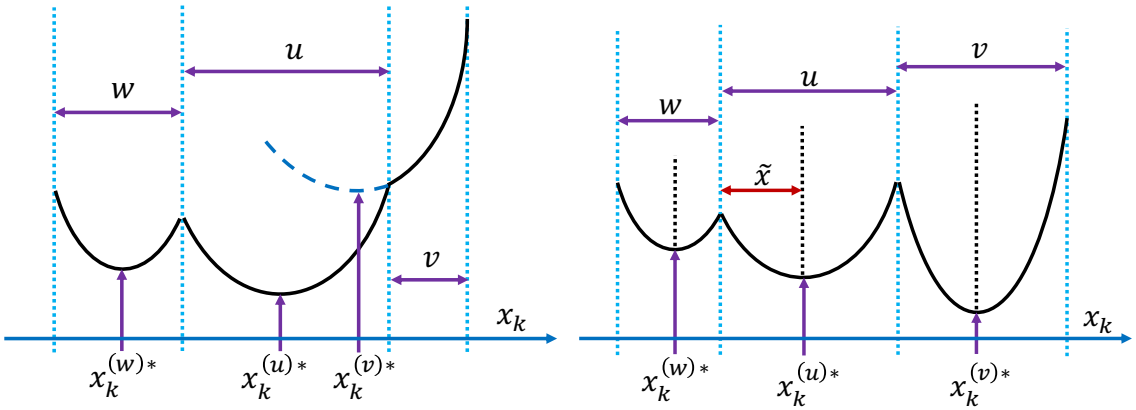
This means that the space of x is divided into regions such that $H_y(x)$ is perfectly approximated by $f_y^{(u)}(x)$ in the particular region u. In other words, the loss function $H_y(x)$ is a piece-wise (or region-wise) quadratic function of x, if we compute the DTW loss as a summation of squared 2-norms, e.g., $\text{dtw}^2(x, y) = ||x_0 - y_0||_2^2 + ||x_1 - y_0||_2^2 + \ldots$. Similarly, if we use the absolute value as the element distance for the functions in the set $\mathcal{F}_y$, then we obtain piece-wise linear function as $H_y(x)$.

This is shown in Figure 4.3a, 4.3b. We perform Monte-Carlo simulations to generate the points and compute their corresponding DTW loss. The length of x is 6, but we only vary the middle two elements after a random initialization and hence can generate the 3-D plots. The length of y is 10. The elements in both x and y are randomly initialized within $[0, 1]$. Figure 4.3a verifies that $H_y(x)$ is piece-wise quadratic using 2-norms, where Figure 4.3b corresponds to the piece-wise linear function.



(A) Quadratic (B) Linear

FIGURE 4.3: Loss function $d = H_y(x)$. (A): $H_y(x)$ approximated by quadratic $f_y(x)$; (B): by linear $f_y(x)$; The curves on the wall are projections of $H_y(x)$ for better illustration.



(A) Analysis case 1 (B) Analysis case 2

FIGURE 4.4: Loss function analysis. (A): Illustration of transitions from u to v, here $f_y^{(v)}$'s stationary point (where $\nabla_{x_k} f_y^{(v)} = 0$) is outside of v; (B): both u and v have bowl-shapes.

Escaping Local Minima

Some recent theoretical work provides proofs for global convergence in non-convex neural network loss functions, e.g., see [23]. In this section, we offer a different perspective for the analysis by exploiting the fact that the global loss function is piece-wise quadratic or linear obtained by a DP process, and the number of regions is bounded by $O(3^{n+l})$ (Lemma 4.4.1). Without loss of generality, we only consider $H_Y(x)$ being piece-wise quadratic. Treating the regions as a collection of discrete states V, where $|V| < 3^{n+l}$, we first analyze the behavior of escaping u and jumping to its neighbor v, for $u, v \in V$, using the standard gradient descent. Without loss of generality, we only look at coordinate k (x_k is of interest). Assume that after DP, a fraction $y_{p:p+q}$ is aligned with x_k. Taking out the items related to x_k, we can write the local quadratic function in u, and its partial derivative with respect to x_k, as

$$f_y^{(u)} = \sum_{j=p}^{p+q}(y_j - x_k)^2 + \sum_{i,j\in\mathcal{U}} I_{ij}(x_i - y_j)^2 \quad \text{and} \quad \nabla_{x_k} f_y^{(u)} = \sum_{j=p}^{p+q} 2(x_k - y_j) \tag{4.5}$$

where $\mathcal{U} = \{i, j | i \neq k, j \notin [p, p+q]\}$, $I_{ij} \in \{0, 1\}$, which is obtained through DP, and i, j satisfy temporal order. Setting $\nabla_{x_k} f_y^{(u)} = 0$ we get the stationary point at $x_k^{(u)*} = \frac{1}{q+1}\sum_{j=p}^{p+q} y_j$.

Without loss of generality, consider the immediate neighbor $f_y^{(v)}$, the same as $f_y^{(u)}$ except for only the alignment of y_{p+q+1}, i.e.,

$$f_y^{(v)} = \sum_{j=p}^{p+q+1}(y_j - x_k)^2 + \sum_{i,j\in\mathcal{V}} I_{ij}(x_i - y_j)^2 \tag{4.6}$$

where $\mathcal{V} = \{i, j | i \neq k, j \notin [p, p+q+1]\}$. The corresponding stationary point is at $x_k^{(v)*}$. Similarly, for the other immediate neighbor w that aligns $\sum_{j=p}^{p+q-1} y_j$, the

stationary point is at $x_k^{(w)*}$.We have

$$x_k^{(u)*} = \frac{\sum_{j=p}^{p+q} y_j}{q+1} \quad , \quad x_k^{(v)*} = \frac{\sum_{j=p}^{p+q+1} y_j}{q+2} \quad , \quad x_k^{(w)*} = \frac{\sum_{j=p}^{p+q-1} y_j}{q} \tag{4.7}$$

Without loss of generality, assume that the three neighbor regions w, u, v are from left to right, i.e., $x_k^1 < x_k^2 < x_k^3$, for $x_k^1 \in w, x_k^2 \in u, x_k^3 \in v$. The three regions corresponding to three local quadratic functions $f_y^{(w)}, f_y^{(u)}, f_y^{(v)}$, and their local minima (or stationary points) $x_k^{(w)*}, x_k^{(u)*}, x_k^{(v)*}$, are illustrated in Figure 4.4a, 4.4b. Note that we are interested in transition $u \to v$, when u's local minimum is not at the boundary (u has a bowl-shape and we want to jump out).

There could be 3 possibilities for the destination (region v). The first one is illustrated in Figure 4.4a, where $x_k^{(v)*}$ is not inside region v, but somewhere to the left. In this case, it is easy to see the global minimum will not be in v since some part in u is lower (u has the bowl-shape due to its local minimum). If jumping to v, the gradient in v would point back to u, which is not the case of interest.

In the second case, both u and v have the bowl-shapes. As shown in Figure 4.4b, the distance between the bottom of two bowls is $d_k^{(u,v)} = x_k^{(v)*} - x_k^{(u)*}$. The boundary must be somewhere in between $x_k^{(u)*}$ and $x_k^{(v)*}$. Since we need to travel from u to v, the starting point $x_k = \tilde{x} \in u$ must be to the left of $x_k^{(u)*}$ (as shown in the red double-arrows region, in Figure 4.4b). Otherwise the gradient at $\tilde{x}$ will point to region w instead of v. To ensure one step crossing the boundary and arrives at v, it needs to travel a distance of at most $(x_k^{(v)*} - \tilde{x})$, because the boundary between u and v could never reach $x_k^{(v)*}$.

For the third case, v does not have the bowl-shape, but $x_k^{(v)*}$ is to the right of v. We can still travel $(x_k^{(v)*} - \tilde{x})$ to jump beyond v. Similar to case 1, the right neighbor

of v (denoted as $v+$) would have a lower minimum if $v+$ has bowl-shape. Even if $v+$ does not have a bowl-shape, the combined region $[v, v+]$ can be viewed as either a quasi-bowl or an extended v, thus jumping here is still valid.

Next, we need to consider the relationship between feasible starting point $\tilde{x}$ and $f_y^{(w)}$'s stationary point $x_k^{(w)*}$. If $x_k^{(w)*}$ is within region w, since $\tilde{x} \in u$, we know that $\tilde{x} > x_k^{(w)*}$. However, there could be cases in which w does not hold $f_y^{(w)}$'s stationary point. If the stationary point $x_k^{(w)*}$ is to the left of region w, then the inequality $\tilde{x} > x_k^{(w)*}$ becomes looser, but still valid. Another case is that when $x_k^{(w)*}$ is to the right side of w. This means w is monotonically decreasing, so we can combine $[w, u]$ as a whole quasi-bowl region u', and let w' be the left neighbor of the combined u'. Therefore, the above analysis on w', u' and v still holds, and we want to jump out u' to v. Hence we arrive at the following theorem.

Theorem 4.4.2. *Assume that the starting point at coordinate k, i.e. $x_k = \tilde{x}$, is in some region u where $f_y^{(u)}$ is defined in Equation 4.5. Let x and y have lengths n and l, respectively, and assume that $l < n$. To ensure escaping from u to its immediate right-side neighbor region, the expected step size $\mathbb{E}[\eta]$ needs to satisfy: $\mathbb{E}[\eta] > \frac{l}{2n}$.*

Proof. As discussed above, we have two cases as follows.

Case 1: First we consider the case that w, u, v are from left to right, with their stationary points $x_k^{(w)*} < x_k^{(u)*} < x_k^{(v)*}$. A standard gradient descent update is $\tilde{x} \leftarrow \tilde{x} - \eta \nabla_{x_k} f_y^{(u)}|_{x_k=\tilde{x}}$. To ensure one step update could make $\tilde{x}$ jump from u to v, we

obtain:

$$\begin{aligned}
&\tilde{x} - \eta \nabla_{x_k} f_y^{(u)}|_{x_k=\tilde{x}} \geq x_k^{(v)*} \\
\Rightarrow\ &\tilde{x} - \eta \sum_{j=p}^{p+q} 2(\tilde{x} - y_j) \geq x_k^{(v)*} \quad \text{(use Equation 4.5)} \\
\Rightarrow\ &(1 - 2\eta(q+1))\tilde{x} + 2\eta(q+1)\frac{1}{q+1}\sum_{j=p}^{p+q} y_j \geq x_k^{(v)*} \\
\Rightarrow\ &(1 - 2\eta(q+1))\tilde{x} \geq x_k^{(v)*} - 2\eta(q+1)x_k^{(u)*} \quad \text{(use Equation 4.7)}
\end{aligned} \tag{4.8}$$

Now we have two possibilities:

Possibility (a): $1 - 2\eta(q+1) > 0$**:**

$$\text{Inequality 4.8} \Rightarrow\ \tilde{x} \geq \frac{x_k^{(v)*} - 2\eta(q+1)x_k^{(u)*}}{1 - 2\eta(q+1)}$$

To guarantee the gradient direction points to neighbor v, the starting point $\tilde{x}$ has to be to the left of $x_k^{(u)*}$, thus:

$$\begin{aligned}
&\frac{x_k^{(v)*} - 2\eta(q+1)x_k^{(u)*}}{1 - 2\eta(q+1)} \leq \tilde{x} < x_k^{(u)*} \\
\Rightarrow\ &x_k^{(v)*} - 2\eta(q+1)x_k^{(u)*} < (1 - 2\eta(q+1))x_k^{(u)*} \\
\Rightarrow\ &x_k^{(v)*} < x_k^{(u)*}
\end{aligned} \tag{4.9}$$

This is contradictory to the assumption that $x_k^{(w)*} < x_k^{(u)*} < x_k^{(v)*}$, and thus is not valid.

Possibility (b): $1 - 2\eta(q+1) < 0$**:**

$$\text{Inequality 4.8} \Rightarrow\ \tilde{x} \leq \frac{x_k^{(v)*} - 2\eta(q+1)x_k^{(u)*}}{1 - 2\eta(q+1)}$$

Due to the fact that $\tilde{x}$ is inside region u, which must be somewhere to the right of $x_k^{(w)*}$ (with the assumption that w has a bowl-shape), we have:

$$\begin{aligned} & \frac{x_k^{(v)*} - 2\eta(q+1)x_k^{(u)*}}{1-2\eta(q+1)} \geq \tilde{x} > x_k^{(w)*} \\ \Rightarrow\ & x_k^{(v)*} - 2\eta(q+1)x_k^{(u)*} < (1-2\eta(q+1))x_k^{(w)*} \\ \Rightarrow\ & x_k^{(v)*} - x_k^{(w)*} < 2\eta(q+1)(x_k^{(u)*} - x_k^{(w)*}) \\ \Rightarrow\ & \eta > \frac{1}{2(q+1)}(\frac{x_k^{(v)*} - x_k^{(w)*}}{x_k^{(u)*} - x_k^{(w)*}}) \end{aligned} \tag{4.10}$$

Recall that q is an integer and $q \geq 0$, thus

$$1 - 2\eta(q+1) < 0 \ \Rightarrow\ \eta > \frac{1}{2(q+1)} \tag{4.11}$$

Also notice that

$$x_k^{(w)*} < x_k^{(u)*} < x_k^{(v)*} \ \Rightarrow\ \frac{x_k^{(v)*} - x_k^{(w)*}}{x_k^{(u)*} - x_k^{(w)*}} > 1 \tag{4.12}$$

Putting Inequalities 4.10, 4.11 and 4.12 together, we obtain $\eta > \frac{1}{2(q+1)}$.

Case 2: here w, u, v are from right to left, and $x_k^{(w)*} > x_k^{(u)*} > x_k^{(v)*}$. This is very similar to Case 1, so we omit the details and provide the final result as

$$\eta > \frac{1}{2(q+1)}(\frac{x_k^{(w)*} - x_k^{(v)*}}{x_k^{(u)*} - x_k^{(v)*}}) \quad \text{and} \quad \frac{x_k^{(w)*} - x_k^{(v)*}}{x_k^{(u)*} - x_k^{(v)*}} > 1 \tag{4.13}$$

We will arrive at the same result: $\eta > \frac{1}{2(q+1)}$.

Note that the length of the pattern x is l and the length of input y is n. As a result, the expected number of elements in y aligned to a single x_i, $i \in [0, l-1]$ should be n/l, i.e. $\mathbb{E}[q] = n/l - 1$. Taking expectation on both sides of the above inequality,

we obtain $\mathbb{E}[\eta] > \frac{1}{2(n/l-1+1)} = \frac{l}{2n}$. □

Corollary 4.4.3. *With the same assumption of Theorem 4.4.2, let pattern x and input y have the same length, i.e., $n = l$. In order to make one step jumping out of local region u, we should have $\mathbb{E}[\eta] > \frac{1}{2}$.*

In other cases, we consider a dataset $Y = \{y_i | y_i \in \mathcal{R}^n, i = 1, \ldots, m\}$. The DTW loss and its full gradient have the summation form, i.e., $H_Y(x) = \sum_{i=0}^{m} H_{y_i}(x)$ and $\nabla_x H_Y(x) = \sum_{i=0}^{m} \nabla_x H_{y_i}(x)$. The updating of x is done via stochastic gradient descent (SGD) over mini-batches, i.e., $x \leftarrow x + \eta \frac{m}{b} \sum_b \nabla_x H_{y_i}(x)$, where $b < m$ is the mini-batch size, and η is the step size. Though the stochastic gradient is an unbiased estimator, i.e. $\mathbb{E}[\frac{m}{b} \sum_b \nabla_x H_{y_i}(x)] = \nabla_x H_Y(x)$, the variance offers the capability to jump out of local minima.

4.5 Streaming DTW Learning

The typical length of a DTW kernel is much shorter than the input data. Aligning the short kernel with a long input sequence, could lead to misleading results. For example, consider the ECG data sequence which consists of several periods of heartbeat pulses, and we would like to let the kernel learn the heartbeat pulse pattern. However, applying an end-to-end DTW, the kernel will align the entire sequence rather than a single pulse period. If the kernel is very short, it does not even have enough resolution and thus finally outputs a useless abstract.

To address this problem, we bring the SPRING [69] algorithm to output the

patterns aligning subsequences of the original input:

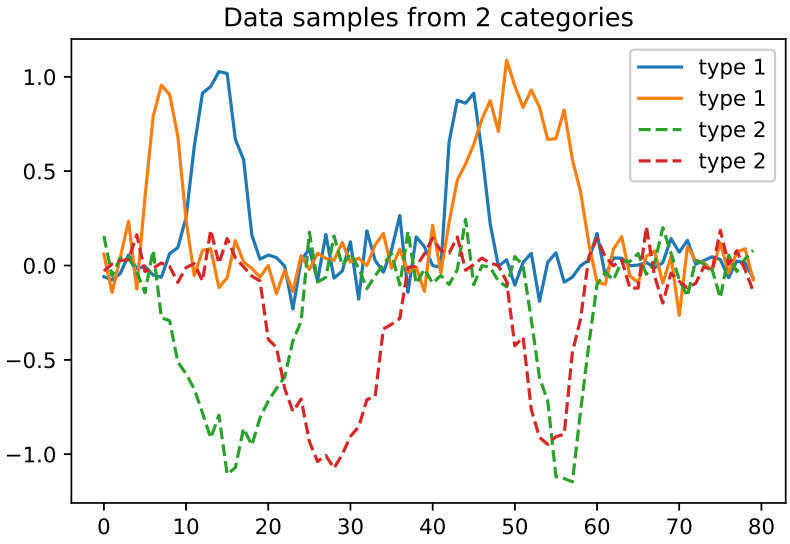
$$x^* = \arg\min_{i,\Delta,x} \mathrm{dtw}^2(x, y_{i:i+\Delta}) \tag{4.14}$$

where $y_{i:i+\Delta}$ denotes the subsequence of y that starts at position i and ends at $i+\Delta$, and x is the pattern (the DTW kernel) we would like to learn. Note that i and Δ are parameters to be optimized.

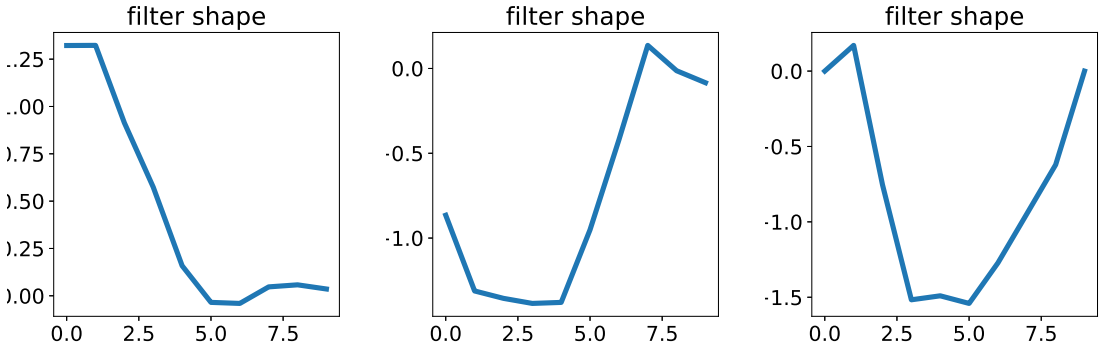
In fact, SPRING not only finds the best matching among all subsequences, but also reports a number of candidate warping paths that have small DTW distances. As a result, we propose two schemes that exploit this property. In the first scheme, we pre-specify a constant k (e.g. 3 or 5) and let SPRING provide the top k best warping paths (k different non-overlapping subsequences that have least DTW distances to the pattern x). In the second scheme, rather than specifying the number of paths, we set a value of ϵ such that all the paths that have distances smaller than $(1+\epsilon)d^*$ are reported, where d^* is the best warping path's DTW distance. After obtaining multiple warping paths, we can do either an averaging, or random sampling as our DTW computing result. In our experiments, we choose $\epsilon = 0.1$ and randomly sample one path for simplicity.

Regularizer in Streaming DTW

Since SPRING encourages the kernel x to learn some repeated pattern in the input sequence, there is no constraint of such patterns' shapes, which could cause problematic learning results. As a matter of fact, some common shapes that do not carry much useful information always occur in the input data. For example, an up-sweep or



(A) Data samples



(B) No reg (C) Week reg (D) Strong reg

FIGURE 4.5: Illustration of the effect of the streaming DTW's regularizer. (A): Data samples. (B): Regularizer with $\alpha = 0$. (C): Regularizer with $\alpha = 1 \times 10^{-4}$. (D): Regularizer with $\alpha = 0.1$.

down-sweep always exists, even the Gaussian noise is a combination of such sweeps. The kernel without any regularization would easily capture such useless patterns and fall into such local minima. To solve this issue, we propose a simple solution that adds a regularizer on the shape of the pattern. Assuming x is of length l, we change the objective to

$$\min_{i,\Delta,x} \ (1-\alpha)\mathrm{dtw}^2(x, y_{i:i+\Delta}) + \alpha||x_0 - x_l|| \tag{4.15}$$

where α is the hyper parameter that controls the regularizer. This essentially forces the pattern to be a "complete" one, in the sense that the beginning and the ending of the pattern should be close. It is a general assumption that we want to capture such "complete" signal patterns, rather than parts of them. As shown in Figure 4.5a, the input sequences contain either upper or lower half circles as the target to be learned. Without regulation, Figure 4.5b shows that the kernel only learns a part of that signal. Figure 4.5c corresponds to a weak regularizer, where the kernel tries to escape from the tempting local minima (these up-sweeps are so widely spread in the input and lead to small SPRING DTW distances). A full shape is well learned with a proper α, as shown in Figure 4.5d. Other shape regularizers could be also used, if they contain prior knowledge from human experts.

4.6 Experiments and Applications

In this experimental section, we compare the proposed scheme with existing approaches. We refer to the end-to-end DTW kernel as **Full DTW**, and the streaming version as **SPRING DTW**. We implement our approach in PyTorch [61].

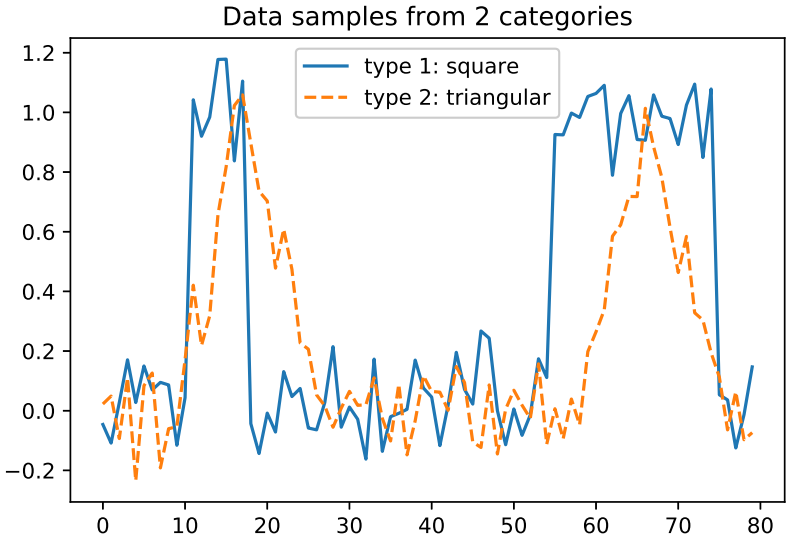
4.6.1 Comparison with Convolution Kernel

Univariate Time Series Experiment

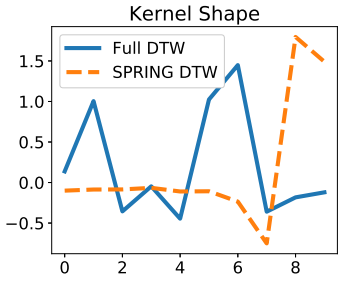
In this very simple classification task, two types of synthetic data sequences are generated. Category 1 only consists of half square signal patterns. Category 2 only has upper triangle signal patterns. Each data sequence was planted with two such signals, but in random locations with random pattern lengths. The patterns do not overlap in each sequence. Also, Gaussian noise is injected into the sequences. Figure 4.6a provides some sample sequences from both categories.

The length of the input sequences is 100 points, where the planted pattern length varies from 10 to 30. There are a total of 100 sequences in the training set, 50 in each category. Another 100 sequences form the testing set, 50 for each type as well. We added Gaussian noise with $\sigma = 0.1$. For comparison, we tested one full DTW kernel, one SPRING DTW kernel, and one convolution kernel. The kernel lengths are set to 10. $\alpha = 0.1$ for SPRING DTW. We append 3 linear layers to generate the prediction.

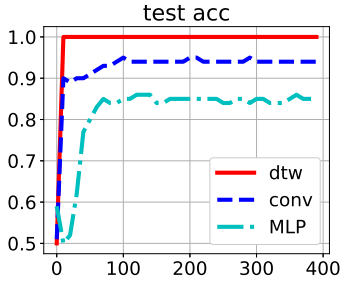
In Figure 4.6b, we show the learned DTW kernel after convergence. As expected, the full DTW kernel tries to capture the whole sequence. Since the whole sequence consists of two planted patterns, the full DTW also has two peaks. On the contrary, SPRING DTW only matches partial signal, thus resulting in a sweep shape. Figure 4.6c and Figure 4.6d show the test accuracy and test loss for 400 iterations. Since both full DTW and SPRING DTW achieve 100% accuracy, and their curves are almost identical, we only show the curve from the full DTW. Surprisingly, the network with the convolution kernel fails to achieve 100% accuracy after convergence on this simple task. The "MLP" represents a network consisting of only 3 linear layers, and performs the worst among all the candidates as expected.



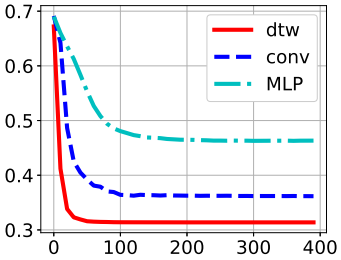
(A) Data samples



(B) Learned kernels



(C) Test acc



(D) Test loss

FIGURE 4.6: Performance comparison on synthetic data sequences (400 iterations)

Multivariate Time Series Experiment

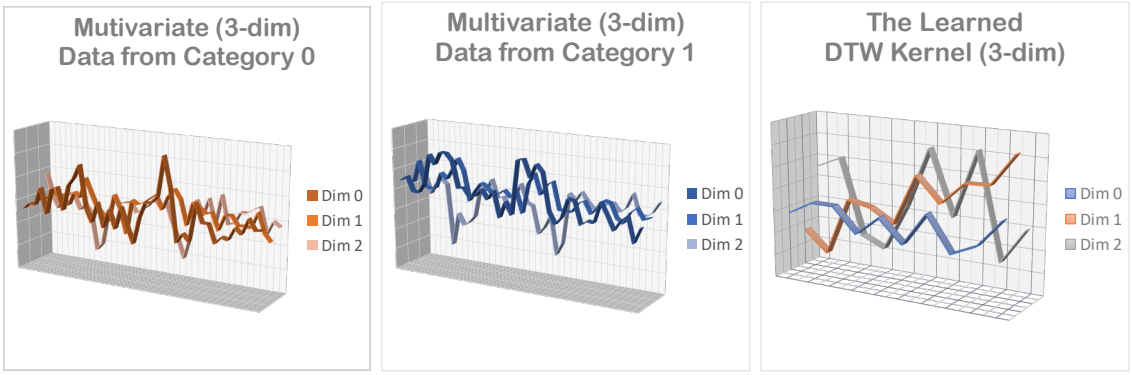
Note that we can easily extend the method to multi-variate time series data (MDTW [73]), without any significant modifications. Multivariate DTWs are often computed in two forms: MDTW-I and MDTW-D [73]. MDTW-I treats each dimension independently, so it is simply a stack of multiple univariate DTWs, thus directly applies to our method. MDTW-D needs to compute multivariate distance $\text{mdtw}^2 = \sum ||\mathbf{x_i} - \mathbf{y_j}||^{\mathbf{2}}, \mathbf{x_i} \in \mathbf{R^m}, \mathbf{y_j} \in \mathbf{R^m}$ in the Dynamic Programming step, instead of the scaler version $\sum ||x_i - y_j||^2$. As long as the norm is well defined, e.g., Euclidean distance, the forward pass and the backpropagation are performed in the same manner. We can even define other distances, as long as their gradients w.r.t. to the vector $\mathbf{x}$ can be computed.

We run a 3-dim multivariate time series classification task here, using MDTW-D and Euclidean distance in our approach. The experiment settings follow Section 4.6.1. The following figures show: one sample data (3-variate series) from each category, the learned kernel, test loss and test acc comparison. Our method (DTW) outperforms others.

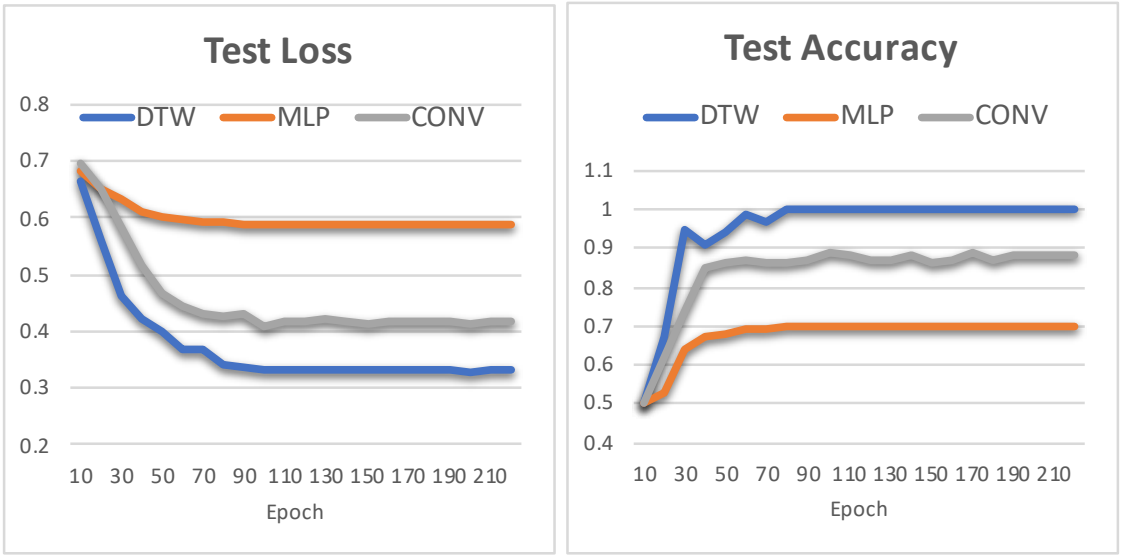
4.6.2 Evaluation of Gradient Calculation

To evaluate the effectiveness and accuracy of the proposed BP scheme, we follow the experimental setup in [17] and perform barycenter computations. The UCR repository [15] is used in this experiment. We evaluate our method against SoftDTW [17], DBA [65] and SSG [17]. We report the average of 5 runs for each experiment. A random initialization is done for all the methods.

The barycenter experiment aims to find the barycenter for the given input se-



(A) Data sample type-0 (B) Data sample type-1 (C) The learned kernel



(D) Test Loss (E) Test Acc

FIGURE 4.7: Multivariate DTWNet Experiment

TABLE 4.1: Barycenter Experiment Summary

	Training Set				Testing Set			
Alg	SoftDTW	SSG	DBA	**Ours**	SoftDTW	SSG	DBA	**Ours**
Win	4	23	21	**37**	11	21	22	**31**
Avg-rank	3.39	**2.14**	2.27	2.2	3.12	2.31	2.36	**2.21**
Avg-loss	27.75	26.19	26.42	**24.79**	33.08	33.84	33.62	**31.99**

quences. We use the entire training set to train the model to obtain the barycenter b_i for each category, and then calculate the DTW loss as:

$$\mathcal{L}_{\text{dtw}} = \frac{1}{N_{\text{class}}} \sum_{i=0}^{N_{\text{class}}} \frac{1}{N_i} \sum_{j=0}^{N_i} \text{dtw}(s_{i,j}, b_i) \tag{4.16}$$

where N_{class} is the number of categories, N_i is the number of sequences in class i, and $s_{i,j}$ is sequence j in class i. The DTW distance is computed using ℓ_2 norm. Clearly, the less the loss, the better is the performance. We also evaluate on the testing set by using $s_{i,j}$ from the testing set. Note that we first run SoftDTW with 4 different hyperparameter settings $\gamma = 1, 0.1, 0.01$ and 0.001 as in [17]. In the training set, $\gamma = 0.1$ outperforms others, while in the testing set, $\gamma = 0.001$ gives the best results, thus we select γ accordingly.

The experimental results are summarized in Table 4.1. "Win" denotes the number of times the smallest loss was achieved, among all the 85 datasets. We also report the average rank and average loss (sum all the losses and divide by number of datasets) in the table. From the results we can clearly see that our proposed approach achieves the best performance among these methods. The details of this experiment can be found in Table 4.2, Table 4.3, Table 4.4 and Table 4.5.

TABLE 4.2: Barycenter Experiment, Average DTW Loss on Training Set, Part 1

	SoftDTW $\gamma = 1$	SoftDTW $\gamma = 0.1$	SoftDTW $\gamma = 0.01$	SoftDTW $\gamma = 0.001$	SSG	DBA	**Ours**
50words	7.337	5.294	5.375	5.478	4.904	4.604	**4.415**
Adiac	0.246	0.218	0.219	0.663	**0.133**	0.136	0.461
ArrowHead	2.713	2.223	1.690	1.979	1.856	**1.555**	1.717
Beef	17.753	9.260	7.889	8.617	17.056	8.618	**7.487**
BeetleFly	34.656	23.368	22.839	23.729	24.771	22.341	**20.544**
BirdChicken	21.017	10.711	9.459	11.982	11.115	12.768	**9.136**
CBF	22.421	14.292	12.891	14.062	11.086	**11.039**	11.592
Car	2.003	1.252	1.063	1.386	**0.940**	1.089	1.230
ChlorineConcentration	24.167	14.339	16.069	15.773	**12.871**	13.227	13.829
CinC_ECG_torso	137.172	99.014	80.749	91.645	80.430	79.076	**69.681**
Coffee	1.056	0.740	1.228	2.038	**0.451**	0.478	0.945
Computers	192.741	198.368	162.193	163.533	**155.350**	164.252	158.226
Cricket_X	45.766	33.312	32.477	33.539	33.537	32.728	**30.582**
Cricket_Y	43.959	31.407	31.865	30.490	31.170	32.640	**29.766**
Cricket_Z	48.030	34.976	34.390	35.690	33.936	34.711	**31.129**
DiatomSizeReduction	0.161	0.146	0.139	0.576	0.061	**0.055**	0.760
DistalPhalanxOutlineAgeGroup	1.791	1.366	1.573	2.406	**1.088**	1.141	1.332
DistalPhalanxOutlineCorrect	2.643	2.081	2.227	2.453	1.896	**1.842**	1.874
DistalPhalanxTW	1.358	1.010	1.031	1.399	**0.691**	0.736	0.937
ECG200	7.365	5.677	7.488	6.931	6.171	6.262	**5.638**
ECG5000	12.282	11.623	13.441	11.390	12.471	11.427	**10.567**
ECGFiveDays	9.987	8.418	7.821	7.001	7.750	**6.927**	11.559
Earthquakes	150.271	**87.765**	91.644	88.571	88.961	88.290	89.370
ElectricDevices	31.243	28.281	28.409	27.694	27.151	27.649	**27.033**
FISH	0.913	0.777	0.639	0.692	**0.487**	0.497	0.789
FaceAll	18.250	15.273	17.053	15.958	13.898	14.416	**13.678**
FaceFour	28.604	24.447	26.018	27.347	30.171	29.341	**22.983**
FacesUCR	16.913	14.094	15.200	15.657	12.979	**12.656**	13.025
FordA	63.812	53.893	55.744	55.351	51.829	53.025	**51.076**
FordB	66.695	56.032	56.154	55.071	52.866	53.447	**51.686**
Gun_Point	7.586	2.525	2.208	2.354	3.393	**2.113**	2.259
Ham	25.753	21.942	19.811	20.530	19.482	20.669	**19.319**
HandOutlines	-	-	-	-	2.094	1.975	2.859
Haptics	19.904	15.475	15.414	16.633	**12.714**	14.710	14.638
Herring	1.778	1.245	1.520	1.526	**0.873**	1.118	1.172
InlineSkate	91.103	34.482	25.691	27.585	30.498	22.163	**21.671**
InsectWingbeatSound	14.798	13.418	13.024	12.506	12.148	12.407	**11.909**
ItalyPowerDemand	2.748	2.317	3.343	2.810	2.222	**2.161**	2.302
LargeKitchenAppliances	125.113	**99.668**	104.094	100.850	109.575	112.017	102.100
Lighting2	84.514	73.412	75.298	74.958	72.800	**71.848**	72.641
Lighting7	32.775	27.390	25.715	26.188	25.786	25.216	**24.230**

TABLE 4.3: Barycenter Experiment, Average DTW Loss on Training Set, Part 2

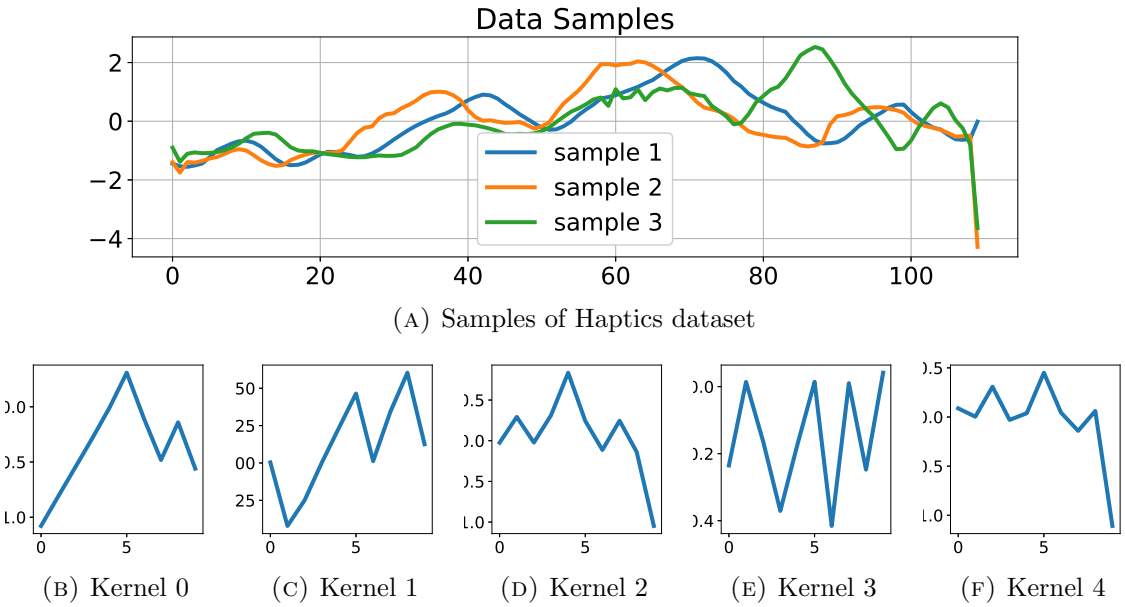
	SoftDTW $\gamma = 1$	SoftDTW $\gamma = 0.1$	SoftDTW $\gamma = 0.01$	SoftDTW $\gamma = 0.001$	SSG	DBA	**Ours**
MALLAT	4.744	5.130	3.332	4.122	2.091	**1.949**	3.461
Meat	0.820	0.492	1.046	1.122	0.040	**0.039**	2.381
MedicalImages	8.110	8.041	8.662	9.552	**6.188**	6.900	6.477
MiddlePhalanxOutlineAgeGroup	0.853	0.748	0.914	1.097	**0.511**	0.511	0.766
MiddlePhalanxOutlineCorrect	0.825	0.652	0.720	1.181	**0.507**	0.508	0.551
MiddlePhalanxTW	0.752	0.584	0.949	1.449	0.416	**0.404**	0.692
MoteStrain	24.706	22.632	**19.159**	20.079	21.629	20.790	20.155
NonInvasiveFatalECG_Thorax1	2.418	2.607	6.163	3.169	**1.139**	1.149	2.852
NonInvasiveFatalECG_Thorax2	2.311	2.169	2.536	2.858	1.088	**1.081**	2.318
OSULeaf	32.206	21.743	21.641	20.844	20.371	19.971	**19.120**
OliveOil	1.217	1.180	1.362	4.419	0.018	**0.017**	1.187
PhalangesOutlinesCorrect	1.681	1.312	1.946	1.684	**1.132**	1.146	1.277
Phoneme	181.389	134.930	135.640	136.674	133.475	121.774	**118.926**
Plane	1.058	0.783	1.202	1.809	0.444	**0.416**	1.075
ProximalPhalanxOutlineAgeGroup	0.620	0.530	0.807	0.946	0.352	**0.336**	0.431
ProximalPhalanxOutlineCorrect	0.859	0.707	1.053	1.164	**0.512**	0.514	0.537
ProximalPhalanxTW	0.630	0.513	0.645	1.089	**0.247**	0.261	0.800
RefrigerationDevices	182.659	156.169	151.285	149.829	152.254	155.795	**137.791**
ScreenType	187.460	156.534	155.496	155.359	**151.867**	158.273	153.550
ShapeletSim	236.166	123.055	124.282	127.652	122.746	123.699	**111.356**
ShapesAll	15.045	8.828	7.745	8.333	8.755	8.929	**7.448**
SmallKitchenAppliances	184.888	177.515	181.719	**177.369**	177.863	181.845	179.879
SonyAIBORobotSurface	8.765	6.722	7.511	7.685	**5.876**	6.252	6.663
SonyAIBORobotSurfaceII	11.896	10.897	12.207	12.315	**9.463**	9.754	11.239
StarLightCurves	16.522	9.449	6.834	6.645	6.557	**6.156**	6.448
Strawberry	2.034	1.642	1.315	1.704	1.225	**1.223**	1.489
SwedishLeaf	2.891	2.124	2.419	2.400	**1.887**	1.936	2.149
Symbols	2.140	1.012	**0.798**	1.212	1.133	0.882	1.573
ToeSegmentation1	35.839	29.387	27.285	26.861	27.618	30.725	**26.201**
ToeSegmentation2	36.316	26.012	23.562	22.410	24.837	24.226	**21.764**
Trace	2.169	2.206	1.249	1.374	**0.767**	0.964	0.981
TwoLeadECG	1.616	1.354	1.900	1.349	**1.015**	1.118	1.084
Two_Patterns	12.811	10.047	8.079	**7.850**	8.528	9.718	8.010
UWaveGestureLibraryAll	77.858	46.754	43.451	42.883	42.470	43.537	**37.055**
Wine	0.738	0.517	0.633	1.109	**0.112**	0.113	0.435
WordsSynonyms	17.235	11.125	10.347	10.832	12.036	11.489	**8.428**
Worms	107.089	73.280	68.799	74.002	75.779	80.812	**60.592**
WormsTwoClass	128.791	86.165	82.996	82.916	83.880	90.286	**77.192**
synthetic_control	16.805	8.880	8.886	9.295	8.924	8.926	**8.328**
uWaveGestureLibrary_X	33.867	20.419	19.155	19.131	18.673	19.393	**17.997**
uWaveGestureLibrary_Y	35.155	19.121	16.982	17.484	18.505	16.598	**15.786**
uWaveGestureLibrary_Z	33.574	19.668	18.401	18.701	18.693	18.248	**16.934**
wafer	30.883	**21.369**	24.101	25.974	23.579	31.298	24.725
yoga	33.428	14.453	14.055	11.822	11.343	12.424	**10.882**

TABLE 4.4: Barycenter Experiment, Average DTW Loss on Testing Set, Part 1

	SoftDTW $\gamma = 1$	SoftDTW $\gamma = 0.1$	SoftDTW $\gamma = 0.01$	SoftDTW $\gamma = 0.001$	SSG	DBA	**Ours**
50words	13.770	11.642	11.031	10.913	11.162	11.195	**10.709**
Adiac	0.838	0.305	0.285	0.585	**0.245**	0.256	0.564
ArrowHead	5.427	4.402	3.690	**3.478**	3.766	3.688	4.009
Beef	14.946	14.178	9.814	11.599	14.822	12.016	**8.813**
BeetleFly	53.030	38.986	**36.598**	38.443	41.506	40.907	36.863
BirdChicken	43.657	**19.897**	22.499	20.594	36.865	30.522	22.585
CBF	24.343	15.210	17.237	15.566	14.194	14.596	**13.755**
Car	4.355	3.072	2.723	2.753	2.654	**2.462**	2.573
ChlorineConcentration	25.474	17.345	18.659	17.919	**16.459**	17.341	16.492
CinC_ECG_torso	180.663	141.548	141.753	132.553	166.838	136.126	**128.023**
Coffee	1.226	0.826	1.293	1.577	0.658	**0.654**	1.045
Computers	176.868	157.962	148.396	153.658	**146.668**	153.739	159.314
Cricket_X	51.207	37.780	38.551	37.916	36.245	37.398	**36.214**
Cricket_Y	43.451	34.059	33.094	33.679	**32.414**	33.642	32.629
Cricket_Z	48.466	36.712	36.351	37.211	36.525	36.586	**35.056**
DiatomSizeReduction	4.010	4.024	3.902	**3.896**	3.899	3.907	4.246
DistalPhalanxOutlineAgeGroup	1.523	1.304	1.526	2.036	**1.061**	1.065	1.841
DistalPhalanxOutlineCorrect	2.785	2.351	2.750	2.557	1.958	**1.956**	1.979
DistalPhalanxTW	1.405	0.997	1.977	1.705	0.806	**0.806**	1.179
ECG200	9.344	7.555	8.911	8.784	7.780	8.422	**6.959**
ECG5000	26.295	22.558	22.592	24.241	25.302	27.389	**22.372**
ECGFiveDays	8.767	10.175	10.589	10.965	7.422	**7.336**	8.647
Earthquakes	156.759	110.492	108.953	109.385	106.678	**102.084**	107.288
ElectricDevices	43.261	37.808	37.130	37.421	36.852	36.939	**35.069**
FISH	1.808	1.661	1.535	1.682	1.464	**1.422**	1.652
FaceAll	20.162	18.590	19.029	20.625	17.536	**17.428**	19.027
FaceFour	38.929	38.416	40.709	40.024	**36.045**	39.897	39.898
FacesUCR	20.339	19.555	20.671	20.296	17.635	**17.090**	18.105
FordA	65.742	56.997	55.973	55.459	53.513	53.965	**52.667**
FordB	69.145	60.406	61.231	59.489	57.921	58.954	**56.693**
Gun_Point	8.030	2.816	2.478	2.428	3.172	2.796	**2.315**
Ham	31.478	28.953	26.317	30.149	26.639	26.455	**25.510**
HandOutlines	-	-	-	-	10.864	10.794	7.733
Haptics	25.431	22.077	20.684	21.966	**17.196**	17.926	24.014
Herring	1.465	1.385	1.331	1.462	**0.915**	0.948	1.372
InlineSkate	127.648	65.537	48.215	45.301	58.181	55.270	**40.608**
InsectWingbeatSound	17.576	16.680	15.102	**15.010**	16.210	15.079	15.493
ItalyPowerDemand	2.523	2.578	2.523	2.820	**2.147**	2.705	2.369
LargeKitchenAppliances	118.456	114.174	**107.120**	107.325	122.776	131.369	110.499
Lighting2	86.450	75.184	**72.139**	81.725	78.196	78.350	73.070
Lighting7	48.326	37.158	**36.557**	38.376	37.673	37.747	41.663

TABLE 4.5: Barycenter Experiment, Average DTW Loss on Testing Set, Part 2

	SoftDTW $\gamma = 1$	SoftDTW $\gamma = 0.1$	SoftDTW $\gamma = 0.01$	SoftDTW $\gamma = 0.001$	SSG	DBA	**Ours**
MALLAT	6.002	4.692	4.717	6.280	3.668	**3.379**	4.938
Meat	0.593	0.259	0.438	1.569	0.041	**0.041**	0.470
MedicalImages	8.252	8.807	9.334	8.741	7.194	**6.916**	7.762
MiddlePhalanxOutlineAgeGroup	0.835	0.770	1.540	1.158	**0.723**	0.733	0.962
MiddlePhalanxOutlineCorrect	1.235	1.223	1.145	2.254	1.187	1.200	**0.998**
MiddlePhalanxTW	0.836	0.729	1.054	1.051	0.592	**0.570**	0.694
MoteStrain	22.970	23.938	22.080	24.643	21.964	**21.094**	25.032
NonInvasiveFatalECG_Thorax1	2.745	2.919	3.560	3.890	1.548	**1.509**	3.561
NonInvasiveFatalECG_Thorax2	2.384	2.888	3.146	3.711	**1.465**	1.521	2.955
OSULeaf	35.193	26.392	23.544	24.177	25.619	**23.188**	23.352
OliveOil	0.961	0.747	2.107	2.002	0.020	**0.020**	1.082
PhalangesOutlinesCorrect	2.067	1.556	1.782	1.487	1.254	**1.239**	1.251
Phoneme	315.058	291.260	**286.486**	286.661	319.069	311.551	289.715
Plane	1.171	0.752	1.247	1.572	**0.534**	0.553	1.109
ProximalPhalanxOutlineAgeGroup	0.595	0.461	0.735	1.230	0.356	**0.351**	0.521
ProximalPhalanxOutlineCorrect	0.704	0.587	0.659	1.020	**0.432**	0.435	0.536
ProximalPhalanxTW	0.775	0.586	0.707	1.699	**0.343**	0.348	0.815
RefrigerationDevices	198.045	167.040	162.204	164.149	164.174	174.493	**156.454**
ScreenType	143.046	**119.853**	126.123	124.484	123.647	136.127	135.216
ShapeletSim	243.103	150.826	151.325	152.915	153.454	153.326	**146.383**
ShapesAll	21.230	15.139	12.812	12.824	14.086	14.084	**12.408**
SmallKitchenAppliances	176.407	173.053	175.462	**171.829**	178.142	173.198	181.316
SonyAIBORobotSurface	8.647	9.489	9.221	10.841	7.459	**7.430**	7.882
SonyAIBORobotSurfaceII	16.137	17.066	16.947	17.336	**14.439**	15.585	15.215
StarLightCurves	17.915	10.925	7.939	7.484	7.457	7.376	**7.316**
Strawberry	2.706	1.921	2.757	2.635	1.609	1.656	**1.599**
SwedishLeaf	2.842	2.442	2.483	2.688	2.087	**2.073**	2.395
Symbols	6.280	5.245	3.973	**3.930**	5.395	4.862	4.498
ToeSegmentation1	43.606	36.174	34.410	35.602	35.703	36.982	**34.158**
ToeSegmentation2	72.831	58.728	**47.650**	51.515	54.558	57.188	53.725
Trace	2.879	1.720	1.374	1.448	**0.818**	0.963	1.048
TwoLeadECG	1.619	1.275	1.441	1.708	**1.185**	1.238	1.383
Two_Patterns	12.546	9.490	8.107	**8.012**	9.502	8.415	8.943
UWaveGestureLibraryAll	78.820	51.054	46.568	47.568	45.660	47.290	**41.052**
Wine	0.807	0.549	2.339	1.202	0.103	**0.101**	0.765
WordsSynonyms	25.751	18.486	17.173	16.387	20.113	19.359	**16.247**
Worms	169.897	111.593	99.132	97.843	123.028	111.755	**94.781**
WormsTwoClass	172.790	119.187	110.483	**105.179**	114.444	113.869	105.696
synthetic_control	17.031	9.794	9.631	9.806	9.620	9.692	**9.126**
uWaveGestureLibrary_X	34.245	21.273	19.248	19.091	20.334	20.586	**18.738**
uWaveGestureLibrary_Y	37.343	20.565	19.212	18.875	19.129	19.481	**17.015**
uWaveGestureLibrary_Z	35.221	22.716	20.444	20.108	20.072	21.296	**18.303**
wafer	30.642	**23.935**	27.221	24.758	30.820	32.328	24.577
yoga	24.982	14.573	15.286	11.849	11.681	11.887	**11.120**



(A) Samples of Haptics dataset

(B) Kernel 0 (C) Kernel 1 (D) Kernel 2 (E) Kernel 3 (F) Kernel 4

FIGURE 4.8: Illustration of DTW Decomposition

4.6.3 Application of DTW Decomposition

In this subsection, we propose an application of DTWNet as a time series data decomposition tool. Without loss of generality, we design 5 DTW layers and each layer has one DTW kernel, i.e., x_i. The key idea is to forward the residual of layer i to the next layer in this network. Note that DTW computation $\text{dtw}(y, x_i)$ will generate the warping path like Equation 4.2, from which we obtain the residual by subtracting the corresponding aligned $x_{i,j}$ from y_j, where j is the index of elements.

Figure 4.8 illustrates the effect of the decomposition. Kernel 0 to kernel 4 correspond to the first layer (input side) till the last layer (output side). The training goal is to minimize the residual of the network's output, and we randomly initialize the kernels before training. We use the Haptics dataset from the UCR repository to demonstrate the decomposition.

After a certain amount of epochs, we can clearly see that the kernels from different

layers form different shapes. The kernel 0 from the first layer, has a large curve that describes the overall shape of the data. This can be seen as the low-frequency part of the signal. In contrast, kernel 4 has those zig-zag shapes that describe the high-frequency parts. Generally, in deeper layers, the kernels tend to learn "higher frequency" parts. This can be utilized as a good decomposition tool given a dataset. More meaningfully, the shapes of the kernels are very interpretable for human beings.

4.7 Conclusions and Future Work

In this chapter, we have applied DTW kernel as a feature extractor and proposed the DTWNet framework. To achieve backpropogation, after evaluating DTW distance via Dynamic Programming, we compute the gradient along the determined warping path. A theoretical study of the DTW as a loss function is provided. We identify DTW loss as region-wise quadratic or linear, and describe the conditions for the step size of the proposed method in order to jump out of local minima. In the experiments, we show that the DTW kernel could outperform standard convolutional kernels in certain tasks. We have also evaluated the effectiveness of the proposed gradient computation and backpropogation, and offered an application to perform data decomposition.

Chapter 5

Speeding Up The Inference

5.1 Introduction

Following the line of research, we are aiming to speed up the computation, especially for the deep neural network model. In this chapter, we provide a model compression technique to achieve this goal. Note that this compression technique is a general approach, which could be applied in a variety of network architectures.

In the past decade, deep neural networks (DNNs) have made remarkable achievements in different AI domains, such as ImageNet challenge, Go game, machine translation, etc. However, the complexity of a DNN has also increased, in order to provide the required capacity for those sophisticated tasks. An over-parameterized network not only requires more and more computation power in both training and inference phases, but also sometimes even hurts the performance of the DNN. In fact, Dropout is the best example to show that a dense network performs much worse than its sparse counterpart, in terms of generalization ability. Exploiting the benefits from sparsity

and capacity, researchers are finally able to achieve a better performance compared to a directly trained dense network.

It is well known that a significant fraction of a modern DNN's parameters are redundant. To identify the redundancy in a network, an exhaustive search in the structure space is believed to be not practical, thus researchers have brought up many ideas to eliminate redundancy without loosing capacity. One typical attempt is knowledge distilling, which could entirely change the original network architecture. The others generally fall into the category of network pruning, where the original model skeleton is kept, but the less significant components are carefully removed. Weight level pruning is typically performed on smaller sized networks such as LeNet, while structured or channel-wise pruning is performed on larger CNNs such as VGG and ResNet. Both distilling and pruning could lead to a better Interpretability and hence have attracted growing research interests in recent years. In this chapter, we address the sparsification problem via structured pruning, and provide an efficient solution obtaining state-of-the-art results.

Related Work

Observing that the network parameters (weights) can be expressed in a matrix form, matrix approximation techniques have been adopted to achieve sparsity. For example, [19] proposed a low-rank matrix factorization to sparsify weights and provide feature predictions. As an extension, tensor decomposition is also utilized in [20] to speed up the training of convolutional neural networks.

Other than factorization approaches, pruning methods have been shown to be more efficient. In [30], the authors pruned the parameters that have smaller values,

and found it can largely sparsify the network while keeping the same classification accuracy. Later in [29], the authors combined pruning with the idea of low-precision representation, to achieve higher compression ratios in terms of storage. Following a Bayesian learning framework, [78] offered a mixture of Gaussians as priors to optimize the loss function and achieved simultaneous pruning and training. In [54], the authors use variational dropout to provide a method called SparseVD that achieves the best known weight sparsification performance, in several network structures. Quantization approaches such as SqueezeNet [32], use a 6-bit representation to obtain 510x reduction of model size of AlexNet. [43] even observed that an extreme 2 or 3 digits representation could be enough to achieve a high accuracy.

To obtain interpretable results, structured pruning attracts more and more attention recently. Typically, structured pruning directly operates on CNN channels or blocks, rather than individual weights. The benefit of preserving the channel structure is to take advantage of modern algebra libraries that apply fast matrix operations. Adding or removing entire channels rather than individual weights could achieve much faster computation in practice. In [82], the authors propose a method called Structured Sparsity Learning (SSL) to regularize the filters in DNNs. A Bayesian approach to prune channels is proposed in [58]. Similarly, [48] uses hierarchical priors and achieve very compact networks compared to previous methods. In [49], the authors adopt an ℓ_0 regularization to effectively remove redundant neurons or channels. To take advantage of convexity, the authors of [31] put masks on the network sub-blocks, and apply ℓ_1 penalty on those masks. To question the common practice that smaller values mean less importance, the authors in [85] proposed a channel pruning approach that does not rely on this belief, and showed it to be also effective in their experiments.

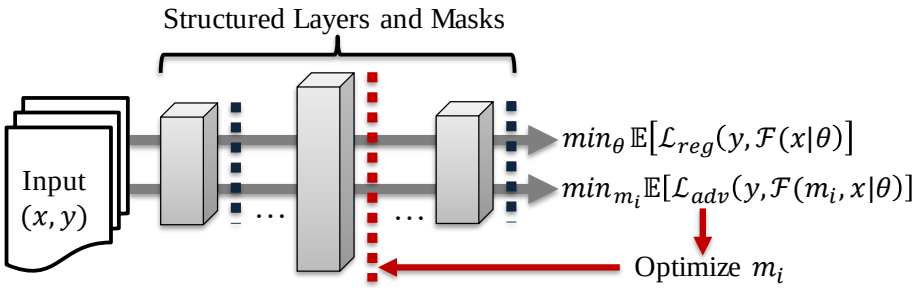


FIGURE 5.1: Adversarial Structured Pruning Diagram

Adversarial example [75] is a small perturbation (usually indistinguishable by human) in the input that leads to erroneous DNN outputs. It has been shown that DNNs have nearly zero defense against adversarial attacks [1]. To generate adversarial samples, many optimization based techniques are proposed such as Carlini & Wagner Attack [12], Elastic-Net Attack [14], etc. The general idea is to apply gradient based approaches to minimize a constraint adversarial loss function, which is designed to fool the network, e.g., output the wrong label. The constraint guarantees that the perturbed input is very close to the original, due to the fact that the larger the perturbation, the easier will it be to find adversarial examples. In [60], adversarial loss is used to construct a novel dropout scheme, and achieve very good accuracies in several models.

Overview of the Proposed Approach

In this chapter, we address the problem of structured sparsification (channel-wise pruning) given a pretrained network. Inspired by adversarial attacks, we employ this idea to identify network components that are sensitive to the adversarial loss. Unlike

a defense method that wants to eliminate such sensitivity, we instead would like to keep them and remove the insensitive parts to achieve sparisity. In contrast to the other pruning methods that do random perturbations on the output, or simply prune small values, the adversarial loss points us to the "best direction", which guides us to preserve the channels that contribute most to the final network output.

It is worth noting that **Adversarial Dropout** [60] shares similar spirits, by applying masks trained on adversarial loss. However, the main difference between our work and Adv-dropout is that: 1) the detailed method is entirely different (illustrated in the next section); 2) we are solving a different problem. Since the storage size is out of interest, quantization methods are not adopted or compared in this chapter.

The key contributions include: We propose a sparsification method that alternates between a regular training step and an adversarial pruning step, in a layer-wise manner. To the best of the authors' knowledge, this is the first attempt to sparsify a network via solving a constrained adversarial optimization problem. We adaptively adjust the constraint hyper-parameters in the adversarial pruning phase for different layers, to achieve the best layer sparsity. We achieve state-of-the-art results in popular models like VGG and ResNet. We empirically analyze the sparsity-accuracy trade-off, and the impact on the adversarial robustness of the pruned network.

5.2 Adversarial Structured Pruning

5.2.1 Notations and Definitions

Let $\mathcal{F} : \mathcal{R}^p \rightarrow \mathcal{R}^q$ be the network model of interest, where p and q are the input and output dimensions. Without loss of generality, we narrow the model for classification

tasks, thus q could be the number of categories. $\mathcal{F}$ is parameterized by θ, and θ consists of all trainable parameters like weights and bias. We denote $x \in \mathcal{R}^p$ as the input, associated with a label $y \in \{1, \ldots, q\}$. (x, y) are pairs from the training set $\mathcal{X}$.

Assume that $\mathcal{F}$ has a total of K structured layers, e.g., Convolution + Batch-Norm + ReLU. Let $w = \{w_i\}, i \in [0, K-1]$ represent the output of each structured layer. For example, in a convolutional layer, $w_{i,j} \in \mathcal{R}^{d \times d}$ could be the channel j's output, which is a feature map of size $d \times d$. **We append a mask to each channel to switch on or off the entire channel.** Formally speaking, for each $w_{i,j}$, we associate a mask $m_{i,j} \in \{0, 1\}$. Then the masked output for layer i's channel j (assuming in total n channels) is written as $o_{i,j} = w_{i,j} \odot \text{broadcast}(m_{i,j}), \quad i \in [0, K-1],\ j \in [0, n]$, where $\odot$ is the Hadamard (element-wise) product, and broadcast() duplicates scalar $m_{i,j}$ to a matrix shaped $d \times d$. The final output of the model can be denoted as $\mathcal{F}(x|\theta)$. Since layer $(i-1)$'s masked output is layer i's input, we can write the layer-wise relationship as: $w_i = \mathcal{F}_i(o_{i-1}|x, \theta),\ i \in [0, K-1]$. This is illustrated in Figure 5.1.

5.2.2 Regular Training Step

The method alternates between a regular training phase and an adversarial pruning phase. For the regular training, just as a standard DNN training over the training set $\mathcal{X}$, we solve

$$\theta = \arg\min_{\theta} \mathbb{E}_{(x,y)\in\mathcal{X}}[\mathcal{L}_{reg}(y, \mathcal{F}(x|\theta))] + \alpha R(\theta) \tag{5.1}$$

where $\mathcal{L}_{reg}$ is the regular training loss, typically being a cross entropy loss for classification tasks. R is the regularizer with a factor of α. This is done via standard

techniques like SGD.

5.2.3 Adversarial Pruning Step

Constructing Adversarial Loss

For an input x, we solve a following problem to get an adversarial sample $x + \delta$:

$$\begin{aligned} &\min_{\delta} \ \mathbb{E}_{(x,y)\in\mathcal{X}}[\mathcal{L}_{adv}(x, \epsilon, \delta|\theta)] \\ &s.t. \ (1-\epsilon)||x|| \leq ||x+\delta|| \leq (1+\epsilon)||x|| \end{aligned} \tag{5.2}$$

where δ is the perturbation on the input, ϵ is the allowable deviation from x's original norm, and typically less than 0.1. Conventionally, it is often the ℓ_∞ norm that is used to ensure that each pixel will not be significantly modified, but ℓ_1 is also used to promote sparsity. $\mathcal{L}_{adv}$ is the adversarial loss that tries to fool the network deviating from the correct label. Here we adopt a simple form: $\mathcal{L}_{adv} = -\mathcal{L}_{reg}(y, \mathcal{F}(x+\delta|\theta))$, which means that we simply want to maximize the regular loss to the correct label, but are not concerned with the distribution of the final softmax output.

Minimizing Adversarial Loss

In the adversarial pruning phase, for layer i, we fix all the other parameters (θ, m_j for $j \neq i$), but only optimize m_i. In particular, we solve

$$\min_{\hat{m}_i \in \mathcal{C}_i} -\mathbb{E}_{(x,y)\in\mathcal{X}}[\mathcal{L}_{reg}(y, \mathcal{F}(\hat{m}_i, x|\theta)] \tag{5.3}$$

where $\mathcal{C}_i$ is the constraint set parameterized by ϵ_i, i.e., $\mathcal{C}_i = \{\hat{m}_i : \ (1-\epsilon)||w_i||_\infty \leq ||\hat{m}_i \odot w_i||_\infty \leq (1+\epsilon)||w_i||_\infty\}$. To solve Equation 5.3, we use the projected gradient descent (PGD) method. Initializing all $m_i = \mathbf{1}$, the PGD's update step is: $\hat{m}_i \leftarrow \text{Proj}_{\mathcal{C}_i}(\hat{m}_i - \eta \nabla_{\hat{m}_i} \mathbb{E}[\mathcal{L}_{adv}])$, where η is the step size. $\text{Proj}_{\mathcal{C}_i}$ operator ensures projection onto the $\mathcal{C}_i$. Due to ℓ_∞ norm, the projection is simplified to be a hard thresholding function applied on each element $\hat{m}_{i,j}$:

$$\text{Proj}_{\mathcal{C}_i}(\hat{m}_i) = \begin{cases} \hat{m}_{i,j}, & 1-\epsilon \leq \hat{m}_{i,j} \leq 1+\epsilon \\ 1 + \text{sign}(\hat{m}_{i,j} - 1)\epsilon, & \text{otherwise} \end{cases} \tag{5.4}$$

Sparsification

After obtaining the updated soft-mask $\hat{m}_i$, we then perform pruning based on $\hat{m}_i$. There will be two cases.

Case 1 happens when the solution of Equation 5.3 hits $\mathcal{C}_i$'s boundaries. More precisely, $\hat{m}_{i,j} = 1 \pm \epsilon, \quad \text{for some } j \in [0, |\hat{m}_i| - 1]$. Note that this can always happen as long as we set ϵ small enough, meaning that we can shrink the constraint set to let the unconstrained solution of Equation 5.3 be outside of $\mathcal{C}_i$. Thus the $\text{Proj}_{\mathcal{C}_i}$ operator would clamp some elements to project back to $\mathcal{C}_i$. In this case, we simply keep the elements that are hitting or very close to the boundaries, and remove the others. So the clipping functions would be

$$m_i = \text{Clip}(\hat{m}_i) = 1 \text{ if } |\hat{m}_{i,j}| \geq 1 + \tau\epsilon; \ \ 0, \text{ otherwise} \tag{5.5}$$

where τ is a threshold parameter. In our experiments we set $\tau = 0.9$. By performing

such clipping, we are removing components that are not sensitive to the adversarial loss.

Case 2 is that the $\text{Proj}_{\mathcal{C}_i}$ operator does not actually perform clamping, so the unconstrained solution is already inside $\mathcal{C}_i$. Thus we need to change the strategy, and only remove the smallest s elements, in a greedy way. For example, in the previous iteration, layer i already removed s_i components, so at this iteration we just try to remove s_i+k_i smallest elements, where k_i is dynamically adjusted starting from 1. The dynamic adjusting of k_i can be described as: $k_i \leftarrow 2k_i$ if performance (test accuracy) did not drop in last iteration, otherwise $k_i \leftarrow k_i/2$. So in case 2, the clipping function is defined as:

$$m_i = \text{Clip}(\hat{m}_i, s) = 0 \text{ if } |\hat{m}_{i,j}| \text{ is within } s \text{ smallest}; \quad 1, \text{ otherwise} \tag{5.6}$$

5.2.4 Putting Together

After obtaining m_i by clipping, to alleviate the problem of accuracy drop due to sparsity, for each layer i, we need to perform a regular training (Equation 5.1) to adjust θ correspondingly, using Adam method. If regular training could not yield a satisfactory performance within T iterations, we reverse the mask to be previous m_i. Usually we adopt a learning rate decaying schedule, e.g., every I ($I < T$) iterations reduce the learning rate to 1/10, for the regular training. Note that the adversarial phase does not need to have any scheduling in practice. Starting from the layer 0 (closest to the input), we perform both training for each layer. After iterating all the layers, we loop back. If all the layers are no longer able to be sparsified, we terminate the entire process.

5.3 Experimental Evaluation

Following the recent papers [58], [85], we experiment on VGG-like and ResNet20. The structured pruning is performed on the channel level. We report the each layer's sparsity, the error rate, the **pruning ratio ρ and FLOPs saving ratio** β. ρ is the dense model's parameter number, divided by the pruned model's parameter number, i.e., $\rho = |\theta_{\text{orig}}|/|\theta_{\text{sparse}}|$, and β is dense model's FLOPs divided by the pruned model's.

We run VGG-like and ResNet20 experiments 5 times each, then report the averaged results. The hyper parameters include: $T = 10, \tau = 0.9$; adv learning rate $\eta_{\text{adv}} = 1e-2$; reg learning rate $\eta_{\text{reg}} = 1e-2$ for VGG-like and $1e-4$ for ResNet20; starting $\epsilon = 1e-3$ for VGG-like and 0.1 for ResNet20. Note that most of them are simply set by default, but ϵ for VGG-like is obtained after several tests. This is because the behavior for earlier and later layers of VGG are very different, where the default $\epsilon = 0.1$ is not suitable for later layers. Such phenomenon of VGG is also reported in [31].

VGG-like on CIFAR10 Dataset

VGGs are known to have a large redundancy, and we follow the literature to test on the 16-layer VGG-like model on CIFAR10 dataset. VGG-like has two linear layers, and takes a parameter k as a factor to control the number of neurons or channels in each layer, where in standard case $k = 1.0$. We evaluate the VGG-like model with $k = 1.0$ and $k = 1.5$ (same as in [58]). The sparsification results are shown in Table 5.1.

From the table we can easily see that the proposed scheme performs much better than the other methods. Especially for the later layers that have a large number of

TABLE 5.1: Sparsity and FLOPs in VGG-like, $k = 1.0$ and $k = 1.5$

$k = 1.0$	Layer Sparsity														Err %	ρ (Params)	β (FLOPs)
Original	64	64	128	128	256	256	256	512	512	512	512	512	512	512	7.2	1.0	1.0
SparseVD [54]	64	62	128	126	234	155	31	81	76	9	138	101	413	373	7.2	3.168	1.946
StructuredBP [58]	64	62	128	126	234	155	31	79	73	9	59	73	56	27	7.5	6.255	2.061
StructuredBPa[58]	44	54	92	115	234	155	31	76	55	9	34	35	21	280	9.0	7.705	2.319
ours 1	32	38	86	86	155	149	100	150	3	2	2	2	18	12	7.9	**10.571**	**2.692**
ours 2	24	48	91	96	193	179	82	40	4	2	3	3	4	7	8.9	**12.464**	**2.485**
$k = 1.5$	Layer Sparsity														Err %	ρ (Params)	β (FLOPs)
Original	96	96	192	192	384	384	384	768	768	768	768	768	768	768	6.8	1.0	1.0
SparseVD[54]	96	78	191	146	254	126	27	79	74	9	137	100	416	479	7.0	4.691	2.575
StructuredBP[58]	96	77	190	146	254	126	26	79	70	9	71	82	79	49	7.2	8.616	2.711
StructuredBPa[58]	77	74	161	146	254	125	26	78	66	9	47	55	54	237	7.9	9.719	2.845
ours	37	56	117	136	274	192	40	3	1	2	1	2	8	4	7.83	**18.747**	**3.199**

TABLE 5.2: Sparsity and FLOPs in ResNet20

Alg	Orig	[85]	**ours**
Group 1	16×6	12-12-6-6-11-11	1-2-1-1-12-10
Group 2	32×6	32-32-28-28-28-28	11-16-20-16-14-15
Group 3	64×6	47-47-34-34-25-25	52-56-47-35-51-18
Err %	8.0	9.1	**9.0**
ρ	1.0	1.593	**1.613**
β	1.0	1.451	**2.055**

channels (512 or 768), the adversarial sparsification method can significantly eliminate redundancy. This is due to the smaller value of adversarial boundary $\epsilon = 10^{-3}$ applied on this model. In our experiments, we also tested a default $\epsilon = 0.1$ and found it does not perform very well. The default $\epsilon = 0.1$ works normally in the early layers, however, we observe that no elements hit the ϵ boundary from the beginning for the later layers. Thus these layers start clipping with Equation 5.6 instead of Equation 5.5, and lose most of the benefit of adversarial pruning. This is why we need to set ϵ to be smaller than the default value, and found 10^{-3} could work well for these later layers.

ResNet20 on CIFAR10 Dataset

In this subsection, we tested on another popular network model ResNet20, on CIFAR10 dataset. We adopt the same model as in [85] and compare the sparsity with their reported results. The ResNet20 includes 3 ResNet groups, where each group consists of 3 blocks, and each block has 2 convolutional layers. The sparsification result is shown in Table 5.2.

Sparsity-Accuracy Trade-off

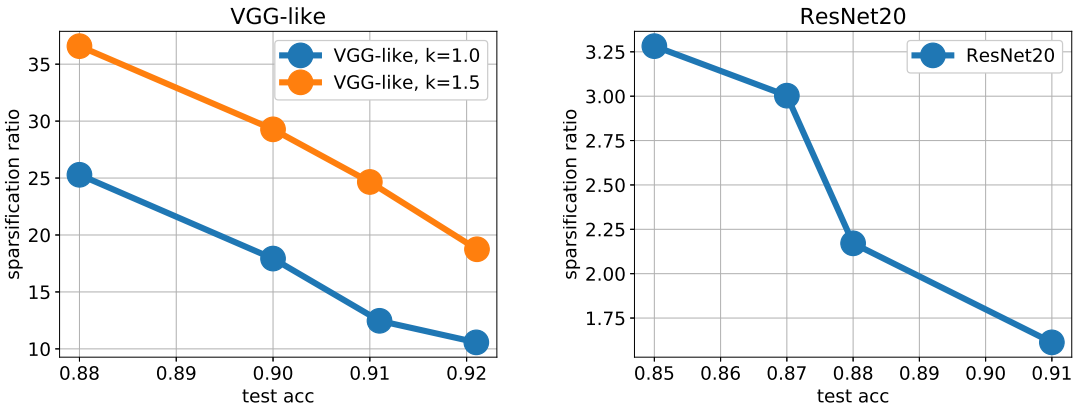
It is well known that the lower the accuracy, the more elements could be safely removed from the network. We preset the accuracy level that the network needs to maintain during the sparsification process, then try to prune as much as possible. Figure 5.2 show the sparsity-accuracy trade-off for VGG-like and ResNet20 networks.

Impact on Adversarial Robustness

Since we employ adversarial attack methods to prune the network and preserve the sensitive channels, it is interesting to evaluate the adversarial robustness of the pruned network. We perform 3 commonly-used attack techniques, FGSM, PGD and DeepFool (from Foolbox package), on the VGG-like ($k = 1.0$) network, to see how attack success rate r changes when the compression ratio ρ increases. The empirical study reveals no significant robustness decay on the pruned network. The attack success rate r remains similar when ρ changes from 4 to 13. Details can be found from Figure 5.3.

5.4 Conclusions and Future Work

In this chapter, we propose a novel Adversarial Structured Pruning method. This approach leverages an iterative process that alternates between a regular training and an adversarial pruning step, in a layer-wise manner. The adversarial pruning step solves a constrained optimization problem where the results are used as a guideline for channel pruning. The components that are insensitive to the adversarial loss, would be removed without sacrificing the model capacity. The experiments reveal



(A) VGG-like (B) ResNet20

FIGURE 5.2: Sparsity-accuracy trade-off

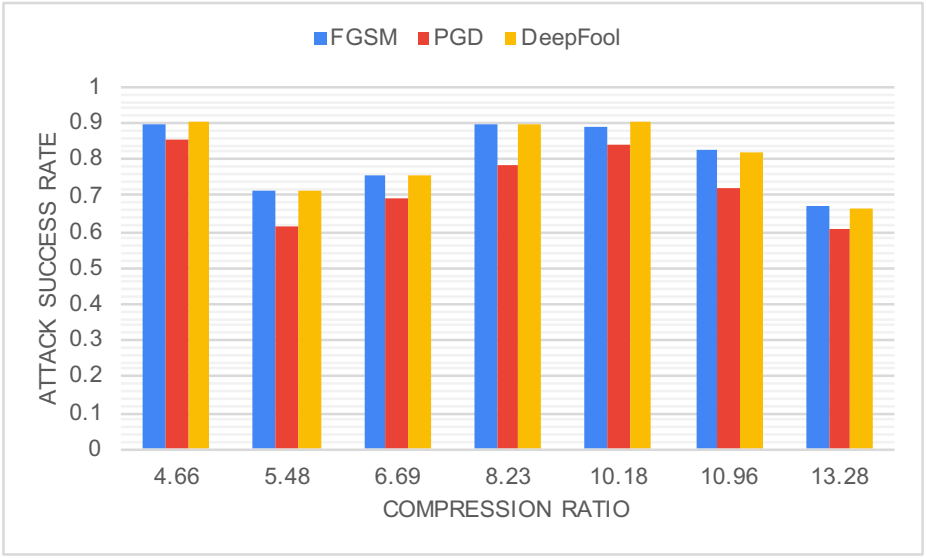


FIGURE 5.3: Compression ratio ρ vs attack success rate r

that this method achieves the state-of-the-art structured pruning performance. The accuracy-sparsity trade-off is empirically studied, which can be used as a guideline in practice. We also carry out a preliminary study on the adversarial robustness of the pruned network, and the initial results do not show any robustness decay after pruning. In future, we can adopt the quantization techniques to further improve the compression rate.

Chapter 6

Conclusions and Future Work

The Closest Pair and related similarity search problems are challenging tasks in the domain of data mining and machine learning. They are also the key steps to dig into the data and find the meaningful results from the data. Generally speaking, this problem requires choosing the proper measurement metric to define the similarity, as well as using proper techniques to identify the target patterns. In this thesis, there are four main chapters that each one address one of the key subproblems in the general Closest Pair problem. These proposed approaches form a set of techniques that could be used in a variety of applications and in different data types. The work has achieved the state-of-the-art performance at the time they published.

The first chapter aims to address the Closest Pair of Points (CPP) problem. Since most of the data could be represented as a vector in a high dimensional space, the CPP solvers could be the baseline as it targets the most general case. We have proposed two approximate algorithms named as ACP-D and ACP-P. They share a similar spirit of converting high dimensional search into 1-D search, which is much

more efficient. We derive the theoretical bounds of the run time for both approaches. The experiments give a fair comparison on every aspect of these algorithms, for a wide range of parameter settings, including data points number n and dimension number m. The proposed ACP-D and ACP-P outputs the target pair with high accuracy at a lower time cost compared with the existing approach, especially in the higher dimensional cases.

The second chapter studies a very closely related problem called the Closest Pair of Subsequences (CPS) problem. The CPS problem is very similar to the original CPP problem, except that the data is no longer independent points but rather numerical sequences. For this particular problem, we proposed a technique called JUMP, that could solve the CPS problem more efficiently than the existing method. JUMP exploits the overlapping parts of consecutive subsequences, and tries to avoid unnecessary computations during the process. It is a deterministic method that guarantees to find the closest pair of subsequences. The pair of subsequences could come from a single sequence, or two different sequences, depending on the tasks. We carried out extensive experiments to evaluate the performance of JUMP on standard benchmark datasets. In most experiments, we can see 10 to 100 times speedups in different datasets, due to a very high skipping of unnecessary computations. However, we also observe that if the skipping fraction of JUMP falls below 50%, there is nearly no improvement over the existing approach. Though this is the rare case but it reveals the limit of JUMP and could be improved in the future work.

The next chapter changes the similarity metric from commonly used Euclidean distance to the Dynamic Time Warping distance, and focus on the time series data types. Inspired by the deep learning domain, we target the problem of pattern mining in time series, through a learnable framework. DTW requires a dynamic programming

process to evaluate, which makes it hard for learning. So we applied DTW in an artificial neural network and proposed a novel approximate method to obtain the gradient of DTW distance. The gradient is used to achieve backpropogation in the neural network. Therefore we can use DTW as both learnable feature extractors and pure distance metric. A theoretical study of the DTW as a loss function is also provided. In the experiments, we show that the learnable DTW kernel could outperform standard convolutional kernels in certain tasks. We have also evaluated the effectiveness of the proposed gradient computation and backpropogation, and offered an application to perform data decomposition through DTW kernels.

In parallel with the work on the neural network, we propose to use pruning techniques to speedup the inference. In the last chapter, we propose a novel Adversarial Structured Pruning method to reduce the network size. The adversarial pruning step solves a constrained optimization problem where the results are used as a guideline for channel pruning. The experiments showed that this method achieves the state-of-the-art structured pruning performance. In addition, we also empirically studied the accuracy-sparsity trade-off, as well as the network robustness decay after pruning.

In the future work, we plan to solve several subsequent problems. Specifically, we would like to perform a better analysis for the JUMP algorithm, without very strong independent identical assumptions on the data elements. Since JUMP itself is still an asymptotically $O(N^2)$ algorithm, it will be very interesting if we can find any deterministic algorithm that runs in sub-quadratic time. For the DTWNet, we are also trying to find a global convergence proof for the method, since the loss function has several unique properties. In addition, we would like to extend this approach to larger data sets and provide a better implementation using advanced GPU computation. Other than these, 2-D warping is another closely related topic

that is worth investigating, since it can generalize our approach to the computer vision or image processing domain, in addition to the 1-D time series.

This line of research in the Closest Pair problem, provides some insights of similarity search in different measurement metrics. Inspired by the recent fast development of deep learning, the conventional search could also be extended to the neural network framework. We can achieve learnable mining on the dataset. Speed improvement is a key research interest. Meanwhile, learning and interpretability are also the focus in the future work.

Bibliography

[1] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[2] P. Beaudoin, S. Coros, M. van de Panne, and P. Poulin, "Motion-motif graphs," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 2008, pp. 117–126.

[3] J. L. Bentley and M. I. Shamos, "Divide-and-conquer in multidimensional space," in *Proceedings of the eighth annual ACM symposium on Theory of computing*. ACM, 1976, pp. 220–230.

[4] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16. Seattle, WA, 1994, pp. 359–370.

[5] X. Cai, A.-A. Mamun, and S. Rajasekaran, "Novel algorithms for finding the closest l-mers in biological data," in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2017, pp. 525–528.

[6] X. Cai, A. A. Mamun, and S. Rajasekaran, "Efficient algorithms for finding the closest l-mers in biological data," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 16, no. 6, pp. 1912–1921, 2018.

[7] X. Cai, S. Rajasekaran, and F. Zhang, "Efficient approximate algorithms for the closest pair problem in high dimensional spaces," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2018, pp. 151–163.

[8] X. Cai, L. Wan, Y. Huang, S. Zhou, and Z. Shi, "Further results on multicarrier mfsk based underwater acoustic communications," *Physical Communication*, vol. 18, pp. 15–27, 2016.

[9] X. Cai, T. Xu, J. Yi, J. Huang, and S. Rajasekaran, "Dtwnet: a dynamic time warping network," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 636–11 646.

[10] X. Cai, J. Yi, F. Zhang, and S. Rajasekaran, "Adversarial structured neural network pruning," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 2433–2436.

[11] X. Cai, S. Zhou, and S. Rajasekaran, "Jump: a fast deterministic algorithm to find the closest pair of subsequences," in *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 2018, pp. 73–80.

[12] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 3–14.

[13] C.-Y. Chang, D.-A. Huang, Y. Sui, L. Fei-Fei, and J. C. Niebles, "D3tw: Discriminative differentiable dynamic time warping for weakly supervised action alignment and segmentation," *arXiv preprint arXiv:1901.02598*, 2019.

[14] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, "Ead: elastic-net attacks to deep neural networks via adversarial examples," *arXiv preprint arXiv:1709.04114*, 2017.

[15] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," July 2015, www.cs.ucr.edu/~eamonn/time_series_data/.

[16] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest pair queries in spatial databases," in *ACM SIGMOD Record*, vol. 29, no. 2. ACM, 2000, pp. 189–200.

[17] M. Cuturi and M. Blondel, "Soft-dtw: a differentiable loss function for time-series," *arXiv preprint arXiv:1703.01541*, 2017.

[18] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.

[19] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *NIPS*, 2013, pp. 2148–2156.

[20] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NIPS*, 2014, pp. 1269–1277.

[21] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen, "A reliable randomized algorithm for the closest-pair problem," *Journal of Algorithms*, vol. 25, no. 1, pp. 19–51, 1997.

[22] T. J. W. I. R. C. R. Division and M. Rabin, *Probabilistic algorithms*, 1976.

[23] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," *arXiv preprint arXiv:1811.03804*, 2018.

[24] S. Fortune and J. Hopcroft, "A note on rabin's nearest-neighbor algorithm," *Information Processing Letters*, vol. 8, no. 1, pp. 20–23, 1979.

[25] Q. Ge, H.-T. Wang, and H. Zhu, "An improved algorithm for finding the closest pair of points," *Journal of computer Science and Technology*, vol. 21, no. 1, pp. 27–31, 2006.

[26] S. Giraldo, A. Ortega, A. Perez, R. Ramirez, G. Waddell, and A. Williamon, "Automatic assessment of violin performance using dynamic time warping classification," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2018, pp. 1–3.

[27] O. Gold and M. Sharir, "Dynamic time warping and geometric edit distance: Breaking the quadratic barrier," *ACM Transactions on Algorithms (TALG)*, vol. 14, no. 4, p. 50, 2018.

[28] A. Gorbenko and V. Popov, "On the longest common subsequence problem," *Applied Mathematical Sciences*, vol. 6, no. 116, pp. 5781–5787, 2012.

[29] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations (ICLR)*, 2016.

[30] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135–1143.

[31] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," *arXiv preprint arXiv:1707.01213*, 2017.

[32] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5mb model size," *arXiv preprint*, 2016.

[33] P. Indyk, M. Lewenstein, O. Lipsky, and E. Porat, "Closest pair problems in very high dimensions," in *ICALP*, vol. 3142. Springer, 2004, pp. 782–792.

[34] M. Jiang and J. Gillespie, "Engineering the divide-and-conquer closest pair algorithm," *Journal of Computer Science and Technology*, vol. 22, no. 4, pp. 532–540, 2007.

[35] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," *Contemporary mathematics*, vol. 26, no. 189-206, p. 1, 1984.

[36] R. J. Kate, "Using dynamic time warping distances as features for improved time series classification," *Data Mining and Knowledge Discovery*, vol. 30, no. 2, 2016.

[37] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and information systems*, vol. 7, no. 3, pp. 358–386, 2005.

[38] S. Khuller and Y. Matias, "A simple randomized sieve algorithm for the closest-pair problem," *Information and Computation*, vol. 118, no. 1, pp. 34–37, 1995.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[40] H. Lei and V. Govindaraju, "Direct image matching by dynamic warping," in *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on.* IEEE, 2004, pp. 76–76.

[41] C.-A. Leimeister and B. Morgenstern, "Kmacs: the k-mismatch average common substring approach to alignment-free sequence comparison," *Bioinformatics*, vol. 30, no. 14, pp. 2000–2008, 2014.

[42] D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern recognition*, vol. 42, no. 9, pp. 2169–2180, 2009.

[43] C. Leng, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," *arXiv preprint*, 2017.

[44] Y. Li, M. L. Yiu, Z. Gong *et al.*, "Quick-motif: An efficient and scalable framework for exact motif discovery," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on.* IEEE, 2015, pp. 579–590.

[45] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[46] J. Lonardi and P. Patel, "Finding motifs in time series," in *Proc. of the 2nd Workshop on Temporal Data Mining*, 2002, pp. 53–68.

[47] M. A. Lopez and S. Liao, "Finding k-closest-pairs efficiently for high dimensional data," 2000.

[48] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," in *NIPS*, 2017.

[49] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint*, 2017.

[50] K. F. Lyon, X. Cai, R. J. Young, A.-A. Mamun, S. Rajasekaran, and M. R. Schiller, "Minimotif miner 4: a million peptide minimotifs and counting," *Nucleic acids research*, vol. 46, no. D1, pp. D465–D470, 2018.

[51] J. Mei, M. Liu, Y.-F. Wang, and H. Gao, "Learning a mahalanobis distance-based dynamic time warping measure for multivariate time series classification," *IEEE transactions on Cybernetics*, vol. 46, no. 6, pp. 1363–1374, 2016.

[52] J. Meng, J. Yuan, M. Hans, and Y. Wu, "Mining motifs from human motion." in *Eurographics (Short Papers)*, 2008, pp. 71–74.

[53] D. Minnen, C. L. Isbell, I. Essa, and T. Starner, "Discovering multivariate motifs using subsequence density estimation and greedy mixture learning," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 22, no. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007, p. 615.

[54] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," *arXiv preprint*, 2017.

[55] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques," *arXiv preprint arXiv:1003.4083*, 2010.

[56] A. Mueen, N. Chavoshi, N. Abu-El-Rub, H. Hamooni, A. Minnich, and J. MacCarthy, "Speeding up dynamic time warping distance for sparse time series data," *Knowledge and Information Systems*, vol. 54, no. 1, pp. 237–263, 2018.

[57] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover, "Exact discovery of time series motifs," in *Proceedings of the 2009 SIAM international conference on data mining*. SIAM, 2009, pp. 473–484.

[58] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," in *NIPS*, 2017, pp. 6778–6787.

[59] P. Xiao, X. Cai, and S. Rajasekaran, "Efficient algorithms for finding edit-distance based motifs," in *International Conference on Algorithms for Computational Biology*. Springer, 2019, pp. 212–223.

[60] S. Park, J.-K. Park, S.-J. Shin, and I.-C. Moon, "Adversarial dropout for supervised and semi-supervised learning," *arXiv preprint arXiv:1707.03631*, 2017.

[61] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[62] S. Pathak and X. Cai, "Ensemble learning algorithm for drug-target interaction prediction," in *2017 IEEE 7th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2017, pp. 1–1.

[63] S. Pathak, X. Cai, and S. Rajasekaran, "Ensemble deep timenet: An ensemble learning approach with deep neural networks for time series," in *2018 IEEE 8th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2018, pp. 1–1.

[64] J. C. Pereira and F. G. Lobo, "An optimized divide-and-conquer algorithm for the closest-pair problem in the planar case," *Journal of Computer Science and Technology*, vol. 27, no. 4, pp. 891–896, 2012.

[65] F. Petitjean and P. Gançarski, "Summarizing a set of time series by averaging: From steiner sequence to compact multiple alignment," *Theoretical Computer Science*, vol. 414, no. 1, pp. 76–91, 2012.

[66] P. A. Pevzner, S.-H. Sze *et al.*, "Combinatorial approaches to finding subtle signals in dna sequences." in *ISMB*, vol. 8, 2000, pp. 269–278.

[67] F. P. Preparata and M. Shamos, *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[68] S. Rajasekaran, S. Saha, and X. Cai, "Novel exact and approximate algorithms for the closest pair problem," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 1045–1050.

[69] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream monitoring under the time warping distance," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 1046–1055.

[70] M. Shah, J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning dtw-shapelets for time-series classification," in *Proceedings of the 3rd IKDD Conference on Data Science, 2016*. ACM, 2016, p. 3.

[71] M. I. Shamos and D. Hoey, "Closest-point problems," in *Foundations of Computer Science, 1975., 16th Annual Symposium on*. IEEE, 1975, pp. 151–162.

[72] Y. Shen, Y. Chen, E. Keogh, and H. Jin, "Accelerating time series searching with large uniform scaling," in *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 2018, pp. 234–242.

[73] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, and E. Keogh, "Generalizing dtw to the multi-dimensional case requires an adaptive approach," *Data mining and knowledge discovery*, vol. 31, no. 1, pp. 1–31, 2017.

[74] P. Steffen, R. Giegerich, and M. Giraud, "Gpu parallelization of algebraic dynamic programming," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2009, pp. 290–299.

[75] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[76] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Efficient and accurate nearest neighbor and closest pair search in high-dimensional space," *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 3, p. 20, 2010.

[77] G. A. ten Holt, M. J. Reinders, and E. Hendriks, "Multi-dimensional dynamic time warping for gesture recognition," in *Thirteenth annual conference of the Advanced School for Computing and Imaging*, vol. 300, 2007, p. 1.

[78] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," *arXiv preprint arXiv:1702.04008*, 2017.

[79] R. Varatharajan, G. Manogaran, M. Priyan, and R. Sundarasekar, "Wearable sensor devices for early detection of alzheimer disease using dynamic time warping algorithm," *Cluster Computing*, pp. 1–10, 2017.

[80] F.-Y. Wang, J. Zhang, Q. Wei, X. Zheng, and L. Li, "Pdp: parallel dynamic programming," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 1, pp. 1–5, 2017.

[81] Z. Wang, A.-A. Mamun, X. Cai, N. Ravishanker, and S. Rajasekaran, "Efficient sequential and parallel algorithms for estimating higher order spectra," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 1743–1752.

[82] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NIPS*, 2016, pp. 2074–2082.

[83] P. Xiao, X. Cai, and S. Rajasekaran, "Ems3: An improved algorithm for finding edit-distance based motifs," in *2018 IEEE 8th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2018, pp. 1–1.

[84] A. C.-C. Yao, "Lower bounds for algebraic computation trees of functions with finite domains," *SIAM Journal on Computing*, vol. 20, no. 4, pp. 655–668, 1991.

[85] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," *arXiv preprint arXiv:1802.00124*, 2018.

[86] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, "Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets," in *IEEE ICDM*, 2016.

[87] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 739–748.